1997 年度 卒業論文

ハードウェア記述言語を用いた行列計算専用プロセッサの設計

学籍番号 9410211

木村・齋藤研　グェン ドゥック ミン

電気通信大学 電気通信学部 電子工学科

指導教官 齋藤 理一郎 助教授

提出日 平成 8 年 2 月 4 日

もくじ

1 序論 **4**

1.1 本研究の目的 4

1.2 ハードウェア記述言語について 4

1.3 本論文の構成 7

2 行列固有ベクトルの計算に用いられるアルゴリズム **8**

2.1 ハウスホルダ変換 8

2.2 二分法 11

2.3 逆反復法 13

2.4 逆変換 15

3 ハードウェア記述言語 VHDL **16**

3.1 基本概念 16

3.1.1 パッケージ宣言 17

3.1.2 エンティティ 17

3.1.3 アーキテクチャ 18

3.1.4 データ型・信号・信号代入文 19

3.1.5 条件つき信号代入文 21

3.1.6 配列 21

3.1.7 列挙型 24

3.2 同時処理文と順序処理文 25

3.2.1 **if** 文 26

3.2.2 **case** 文 27

3.2.3 **loop** 文 29

3.2.4 **wait** 文 29

3.3 解決関数とデルタ遅延 30

3.3.1 解決関数・解決データ型 30

3.3.2 デルタ遅延 33

4 MAX+plus II の使用方法 35

4.1 環境設定 36
4.2 MAX+plus II の起動 36
4.3 新しいプロジェクトの作成 37
4.4 VHDL ソースの作成 38
4.5 VHDL ソースのコンパイル 39
4.6 シミュレーション 41
4.7 書き込み用ファイルの作成 44
4.8 まとめ 46

5 PeakVHDL&FPGA の使用方法 47

5.1 VHDL ソースの作成 47
5.2 VHDL ソースのコンパイル 50
5.3 テストベンチの作成 51
5.4 シミュレーション 52
5.5 論理合成 55
5.6 MAX+plus II とのインターフェイス 56

6 VHDL による行列個有ベクトル計算モデル 58

6.1 ソフトウェア的なアルゴリズムの VHDL 記述 58
6.2 行列固有ベクトルの計算モデル 61
6.2.1 構成について 61
6.2.2 動作について 63

7 シミューレション結果と残された課題 65

A VHDL による行列固有ベクトル計算モデル 68

A.1 2 乗根の VHDL ソース 68
A.2 行列固有ベクトルの計算モデル (本体) 69

B FLEX EPF8282 ボードの回路図 83

C FLEX EPF8282 の書き込みプログラム **84**

1　序論

1.1　本研究の目的

行列固有ベクトルを求める問題はフーリエ変換と並んで，ほとんどの科学技術計算に現れる数値計算問題の１つである．しかし，計算回数が行列の次数の３乗に比例するため，次数が数千にも及ぶ実際の数値計算では実用に耐えないほど非常に多くの時間を必要とする．

この計算時間を短縮しようと，これまで並列化コンピュータを用いたソフトウェア的な手法が試みられてきたが，やはり限界があり，それほどの改善は見込まれない．その原因として，現在使われているワークステーションなどは，どの問題にも対処できるように設計されているため，汎用性には優れているものの，個々の問題に対しては必ずしも最大限の性能を発揮することができない．そこで解決方法として，専用計算機が考えられる．本研究の目的は，行列固有ベクトルの計算に最適な計算機構成を考え，その構成をハードウェア記述言語 VHDL で記述し，シミュレーションを行なうことにより，どうのような計算機構成が適当で，どれほどの時間短縮が見込めるかを見究めることである．

1.2　ハードウェア記述言語について

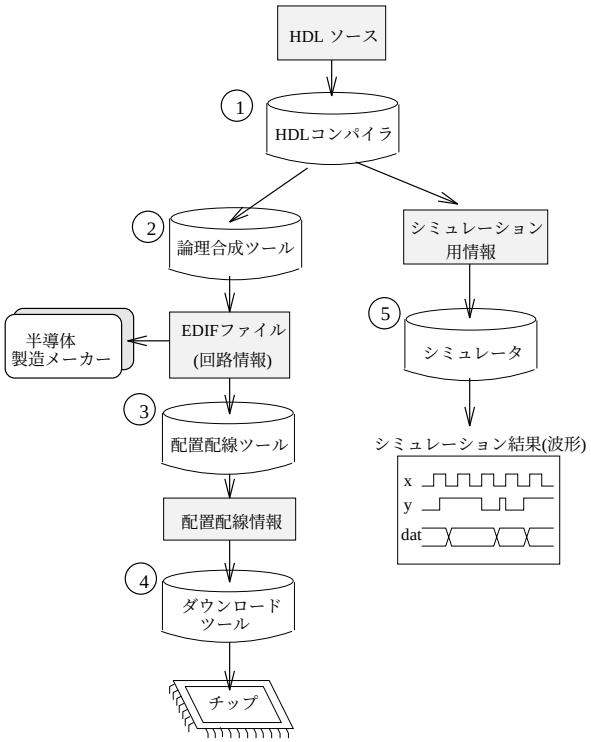
前述したように本研究はハードウェア記述言語 VHDL を使って専用計算機の設計・検証を行なうが，以下はこの VHDL の概要と VHDL を用いた設計手法について簡単に述べる．

近年，ディジタル回路の設計にハードウェア記述言語 (Hardware Description Language,HDL) が広く使われるようになってきた．ハードウェア記述言語は，設計したいもの (回路) を” 文書” で記述し，後はコンパイラに自動変換させるという点では C や Pascal 等のソフトウェア的な言語と同様である．しかし，ハードウェア記述言語とソフトウェアプログラミング言語の間に顕著な違いがある． C や Pascal 等の場合は，コンパイラによって自動的に生成されるのは，コンピュータ上で動作する機械語プログラムで，この機械語プログラムの動作順序がほとんど書かれた” 文書” の順序で決まる．それに対して，ハードウェア記述言語の場合は，生成されるのは” 回路”

を表す情報であり，この回路の動作順序は” 文書 ” で記述した順序と直接的に関係ない．

ハードウェア記述言語には色々な種類があるが，一般的に使われているのは IEEE の規格として定められた VHDL と， Gateway 社によって開発され業界標準として普及した VerilogHDL の 2 つである．当研究室では VHDL を採用している．

VHDL は，米国国防省において， 1970 年代発足した VHSIC (Very High Speed Integrated Circuit) プロジェクトの一環として，ハードウェアを記述する標準言語として 1981 年に提案された． VHDL の V はこの VHSIC プロジェクトの名前に由来している． VHDL は 1987 年に IEEE によって IEEE-1076 仕様として標準化された．



HDL コンパイラを使って HDL ソースからシミュレーションを行なうために必要な情報を生成することができる．更に，シミュレータにより，記述した回路の動作をコンピュータ上で検証することができる．

一方，論理合成ツールを通すと EDIF ファイルが作られる． EDIF(Electronic Design Interchange Format) とは，ディジタル回路を表す業界標準フォーマットである． EDIF ファイルの中に回路を実際のデバイスで実装するために必要な (機能的な[2]) 情報が全て含まれている．

シミュレータが回路の中で起こる現象の全てをカバーすることができないため，シミュレーション段階で得られる結果と，回路の実際の動作の間にしばしば相違がある．従って， EDIF ファイルにもとづいて集積回路を製造した後，その集積回路に対してテストを行なわない限り設計を完全に検証することができない．しかし，実際に集積回路を製造すると高いコストと多くの時間がかかる．そこで，当研究室では FPGA による検証方法を選択している．

FPGA(Field Programmable Gate Array) は特集な集積回路で，その中に，フリップフロップ， and, or，加算器等の基本的な回路素子が数百から数万個入っている．これらの回路素子の相互接続情報は内蔵 RAM に保存されている．外部から適当な電気信号を加え，内蔵 RAM の内容を書き換えることによって簡単に FPGA の機能を変えることができる (外部から信号を加え， FPGA の機能を設定することをコンフィギュレーションという)． FPGA は基板にあったままで何回も書き換えることができるという特徴があり，設計した回路を検証するには最適である．

EDIF ファイルから FPGA に書き込む情報を生成するために配置配線ツールが必要である．また，パソコンを使って FPGA のコンフィギュレーションを行なう場合は更に，パソコンからデータを FGPA に送り出すダウンロードツールも必要である．

現時点で，当研究室で使用可能な VHDL 処理系は **PeakVHDL&FPGA** と **MAX+plus II** の 2 つである． **PeakVHDL&FPGA** は図 1の (1),(2),(5) のモジュールを含んでる． **MAX+plus II** は同図の (1),(2),(3),(5) を含んでる．ダウンロードツール (4) は自作したもの (付録 Cを参照) を使っている．

[2]時間遅延，最高動作周波数等は含まれていない．これらの情報は実際の製造技術に依存するからである．

1.3　本論文の構成

まず第 2章で行列固有ベクトルの計算に使われるアルゴリズムについて説明する．次に，第 3章でハードウェア記述言語について述べる．第 4章と第 5章で，シミュレーションを行なうのに使った 2 つの VHDL 処理システム (**MAX+plus II** と **PeakVHDL&FPGA**) について，具体的な例で使用方法を説明する．最後の第 6章で，本研究が作成した VHDL による行列固有ベクトル計算モデルと得られたシミュレーション結果について述べる．

表記について

本論文は以下の表記方法を使用している．

- *Italic style*
  ウィンドウタイトル, ボタン名やメニュー項目等
- **Bold face**
  ハードウェア記述言語の予約語 (**architecture** や **process** 等)
- `Typewrite`
  端末に出力される情報，ユーザーが入力するテキスト， VHDL のソース (VHDL の文法を説明する時は **bold face** を使用するが，ソースの中ではこの書体を使う).

2　行列固有ベクトルの計算に用いられるアルゴリズム

A が対称行列で，Q が直交行列[3]とする．

$$B = Q^{-1}AQ \tag{1}$$

なる，A と相似な行列 B は以下の性質を持っていることが知られている．

1. B は A と同じ固有値を有する．

2. 共通固有値 λ に対応する A と B の固有ベクトルをそれぞれ λ_A, λ_B とすれば，

$$\lambda_A = Q\lambda_B \tag{2}$$

　が成り立つ．

一方，三重対角行列の固有値と固有ベクトルは二分法と逆反復法により簡単に求めることができるので，B が三重対角行列になるように式 (1) の行列 Q をうまく選べば，任意の対称行列の固有値と固有ベクトルを計算できる．

2.1　ハウスホルダ変換

ハウスホルダ変換は一般の対称行列を三重対角行列に変換するアルゴリズムである．$n \times n$ 行列 A のの第一列 $\mathbf{a}$ を

$$\mathbf{a}^T = (a_1, a_2, ..., a_n) \tag{3}$$

とする．列ベクトル $\mathbf{w}$ と $n \times n$ 行列 Q を次のように定義する．

$$\mathbf{w}^T = (0, a_2 + s, a_3, ..., a_n) \tag{4}$$

$$Q = \mathbf{I} - c\mathbf{w}\mathbf{w}^T \tag{5}$$

但し，s と c は

$$s = \sum_{j=2}^{n} a_j^2 \tag{6}$$

[3] $Q^T = Q^{-1}$ を満たす行列

$$\begin{aligned} c &= \frac{2}{\mathbf{w}^T\mathbf{w}} \\ &= \frac{2}{(0, s+a_2, a_3, ..., a_n)(0, s+a_2, a_3, ..., a_n)^T} \\ &= \frac{2}{(s+a_2)^2 + {a_3}^2 + ... + {a_n}^2} \\ &= \frac{2}{s^2 + 2sa_2 + {a_2}^2 + {a_3}^2 + ... + {a_n}^2} \\ &= \frac{2}{s^2 + 2sa_2 + s^2} \\ &= \frac{1}{s^2 + sa_2} \end{aligned} \tag{7}$$

を満たすスカラである.

Q の定義 (式 (5)) より,

$$\begin{aligned} Q^T &= (\mathbf{I} - c\mathbf{w}\mathbf{w}^T)^T \\ &= \mathbf{I}^T - (c\mathbf{w}\mathbf{w}^T)^T \\ &= \mathbf{I} - c(\mathbf{w}^T)^T\mathbf{w}^T \\ &= \mathbf{I} - c\mathbf{w}\mathbf{w}^T \\ &= Q \end{aligned} \tag{8}$$

$$\begin{aligned} QQ &= (\mathbf{I} - c\mathbf{w}\mathbf{w}^T)(\mathbf{I} - c\mathbf{w}\mathbf{w}^T) \\ &= \mathbf{I}\mathbf{I} - \mathbf{I}(c\mathbf{w}\mathbf{w}^T) - (c\mathbf{w}\mathbf{w}^T)\mathbf{I} + (c\mathbf{w}\mathbf{w}^T)(c\mathbf{w}\mathbf{w}^T) \\ &= \mathbf{I} - 2c\mathbf{w}\mathbf{w}^T + c^2\mathbf{w}(\mathbf{w}^\mathbf{T}\mathbf{w})\mathbf{w}^\mathbf{T} \end{aligned}$$

c の定義式 (7) より, $c = \dfrac{2}{\mathbf{w}^T\mathbf{w}}$ であるから ($\mathbf{w}^T\mathbf{w}$ はスカラ, $\mathbf{w}\mathbf{w}^T$ は $n \times n$ 行列であることに注意),

$$\begin{aligned} &= \mathbf{I} - 2\frac{2}{\mathbf{w}^\mathbf{T}\mathbf{w}}\mathbf{w}\mathbf{w}^T + (\frac{2}{\mathbf{w}^T\mathbf{w}})^2(\mathbf{w}^T\mathbf{w})(\mathbf{w}\mathbf{w}^T) \\ &= \mathbf{I} - 2\frac{2}{\mathbf{w}^T\mathbf{w}}\mathbf{w}\mathbf{w}^T + \frac{4}{\mathbf{w}^T\mathbf{w}}\mathbf{w}\mathbf{w}^T \\ &= \mathbf{I} \end{aligned} \tag{9}$$

つまり, Q が直交対称行列であることが分かる. 更に, $\mathbf{w}\mathbf{w}^T$ の第一列がゼロベクトル $\mathbf{0}$ であるから, $Q = \mathbf{I} - \mathbf{w}\mathbf{w}^T$ の第一列は単位ベクトルとなる.

$$\mathbf{e}^T = (1, 0, 0, ..., 0) \tag{10}$$

従って，B の第一列を $\mathbf{b}$ とすれば

$$\mathbf{b} = Q^{-1}\times(A\times Q\text{の第一列}) \ = Q^{-1}\times(A\times\mathbf{e}) = Q^{-1}\mathbf{a} \tag{11}$$

$Q^{-1} = Q$ なので,

$$\begin{aligned}
\mathbf{b} &= Q\mathbf{a} \\
&= (\mathbf{I} - c\mathbf{w}\mathbf{w}^T)\mathbf{a} \\
&= \mathbf{I}\mathbf{a} - c\mathbf{w}\mathbf{w}^T\mathbf{a} \\
&= \mathbf{a} - c\mathbf{w}(\mathbf{w}^T\mathbf{a}) \\
&= \mathbf{a} - c\mathbf{w}(0, s+a_2, a_3, ..., a_n)(a_1, a_2, ..., a_n)^T \\
&= \mathbf{a} - c\mathbf{w}\{(s+a_2)a_2 + {a_3}^2 + ... + {a_n}^2\} \\
&= \mathbf{a} - c\mathbf{w}(s^2 + sa_2) \\
&= \mathbf{a} - \frac{1}{s^2+sa_2}\mathbf{w}(s^2+sa_2) \\
&= \mathbf{a} - \mathbf{w} = (a_1, -s, 0, 0, ..., 0)
\end{aligned} \tag{12}$$

このように，式 (5) で定義される行列 Q を使って，相似変換 (1) を行なうと，得られる行列 B の第一列，従って第一行 (B が対称であるから) の 3 番目以降の成分が全て 0 になる．この計算を $n-2$ 回繰り返せば，A が図 2のように三重対角行列に変換されるわけである (図 2の灰色の部分は一般的に 0 でない成分を，白い部分はハウスホウルダ変換によって 0 に変換された成分を表している）．

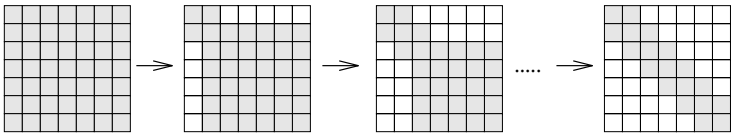


図 2: ハウスホルダ変換の説明図 [4]

次数 n が数千にもおよぶ実際の数値計算では式 (1) と (5) で計算すると $n\times n$ 行列 Q を保存するための膨大なメモリが必要になるので，中間的なスカラ s [5]と列ベクト

[4]ファイル名 = ./fig/househld.xfig.eps

[5]付録 A.2の VHDL ソースの中では識別子の重複を避けるために s の変わりに tmp という名前を使っている.

ル $\mathbf{p}$, $\mathbf{q}$ を求めてから B を計算する.

$$\mathbf{p} = cA\mathbf{w} \tag{13}$$

$$s = \frac{c}{2}\mathbf{p}^T\mathbf{w} \tag{14}$$

$$\mathbf{q} = \mathbf{p} - s\mathbf{w} \tag{15}$$

$$B = A - (\mathbf{w}\mathbf{q}^T - \mathbf{w}\mathbf{q}^T) \tag{16}$$

上記の式 (13) 〜 (16) で求められる B は式 (1) と (5) の B と同じであるのを簡単に確かめることができる.

2.2　二分法

ハウスホルダ変換により，行列 A と相似な三重対角行列

$$B = \begin{pmatrix} \alpha_1 & \beta_1 & & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ 0 & & & \beta_{n-1} & \alpha_n \end{pmatrix},$$

が得られる. B に対応して $\lambda\mathbf{I} - B$ を考える.

$$\lambda\mathbf{I} - B = \begin{pmatrix} \lambda - \alpha_1 & -\beta_1 & & & 0 \\ -\beta_1 & \lambda - \alpha_2 & -\beta_2 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & -\beta_{n-2} & \lambda - \alpha_{n-1} & -\beta_{n-1} \\ 0 & & & -\beta_{n-1} & \lambda - \alpha_n \end{pmatrix}.$$

k 次の多項式 $P_k(\lambda)$ を以下のように定義する.

$$P_k(\lambda) = \det \begin{pmatrix} \lambda-\alpha_1 & -\beta_1 & & & 0 \\ -\beta_1 & \lambda-\alpha_2 & -\beta_2 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & -\beta_{k-2} & \lambda-\alpha_{k-1} & -\beta_{k-1} \\ 0 & & & -\beta_{k-1} & \lambda-\alpha_k \end{pmatrix}, \quad k = 1, 2, ..., n.$$

更に，便宜的に

$$P_{-1}(\lambda) = 0, P_0(\lambda) = 1$$

と置くと，漸化式

$$P_k(\lambda) = (\lambda - \alpha_k)P_{k-1}(\lambda) - \beta_{k-1}^2 P_{k-2}(\lambda),$$

を得る． $k = 1$ の時は

$$P_1(\lambda) = \lambda - \alpha_1$$

となり，また $k = n$ の時は

$$P_n(\lambda) = \det B$$

となる

Sturm 定理によると， λ を固定して，数列

$$P_n(\lambda), P_{n-1}(\lambda), ..., P_1(\lambda), P_0(\lambda) = 1$$

の左から右へ見ていった時，符合の変更回数を $N(\lambda)$ とすると， $N(\lambda)$ は λ より大きい固有値の個数に等しい．従って，三重対角行列 B が重複しない n 個の固有値

$$\lambda_1 < \lambda_2 < ... < \lambda_n$$

を持ち，

$$N(a) \geq n - j, \quad N(b) < n - j$$

を満足するような 2 つの実数 $a, b(a < b)$ が存在すれば，小さい方から j 番目の固有値 λ_j は (a, b) 区間内に存在する．この範囲を十分に狭くすることができるならば，

λ_k の近似値が求められたことになる．区間の幅を半分にしながら，次第に λ_k の存在する範囲を狭くしていく方法が，二分法 (bisection method) である．

a, b の初期値は Gerschgorin 定理に基づいて定めることができる．

$$-a = b = \max_{1 \le i \le n} \{|\beta_{i-1}| + |\alpha_i| + |\beta_i|\}, \quad \beta_0 = \beta_n = 0 \tag{17}$$

また，符合の変化回数を判定するには隣合う 2 つの数値に対して割算を実行すればよい．

2.3　逆反復法

$n \times n$ 行列 B の絶対値最大の固有値 λ_{max} に縮退がない場合は，任意の列ベクトル $\mathbf{x}$ に対して，

$$\lambda_{max}\text{に対応する固有ベクトル} \quad \mathbf{v}_{max} = \lim_{n \to \infty} B^n \mathbf{x} \tag{18}$$

が成り立つことが知られている．

一方，B の固有値 (一般的に絶対値最大でないもの)λ_i に対応する固有ベクトルを $\mathbf{v}_i$ と置けば, 任意のスカラー μ に対して

$$(\mu \mathbf{I} - B)\mathbf{v}_i = \mu(\mathbf{I}\mathbf{v}_i) - B\mathbf{v}_i = \mu \mathbf{v}_i - \lambda_i \mathbf{v}_i = (\mu - \lambda_i)\mathbf{v}_i$$

が成り立つ．この式の両辺に $(\mu \mathbf{I} - B)^{-1}$ をかけると，

$$\mathbf{v}_i = (\mu - \lambda_i)(\mu \mathbf{I} - B)^{-1} \mathbf{v}_i$$

よって，

$$(\mu \mathbf{I} - B)^{-1} \mathbf{v}_i = \frac{1}{\mu - \lambda_i} \mathbf{v}_i \tag{19}$$

行列 B が重複のない n 個の固有値

$$\lambda_1 < \lambda_2 < \ldots < \lambda_n$$

を持つとする．固有値 λ_i に対して

$$|\lambda_i - \mu| < |\lambda_j - \mu|, \quad j \neq i$$

を満たすような実数 μ(他の固有値に比べて，λ_i が最も近いような μ) を選べば，式 (19) により，$\dfrac{1}{\mu-\lambda_i}$ と $\mathbf{v}_i$ がそれぞれ行列 $(\mu\mathbf{I}-B)^{-1}$ の絶対値最大の固有値とその固有値に対応する固有ベクトルになる．従って，任意のベクトル $\mathbf{x}$ から出発して，

$$\mathbf{x}_{k+1}=(\mu\mathbf{I}-B)^{-1}\mathbf{x}_k \tag{20}$$

を繰り返して計算すれば，B の λ_i に対する固有ベクトル $\mathbf{v}_i$ を求めることができる．実際の計算では，逆行列 $(\mu\mathbf{I}-B)^{-1}$ を求める変わりに，n 次連立方程式

$$(\mu\mathbf{I}-B)\mathbf{x}_{k+1}=\mathbf{x}_k \tag{21}$$

を解く．

ここでは証明しないが，$i=1$〜$n-1$ に対して

$m_i=\dfrac{\beta_i}{\alpha_i}$; ($\mu\mathbf{I}-B$ の対角成分を α とする)

$\alpha_{i+1}=\alpha_{i+1}-m_i\times\beta_i$;

を実行すれば，$\mu\mathbf{I}-B$ を以下のように，下三角行列 L と上三角行列 U との積に分解できる (L も U も，主対角成分と片方の副対角成分以外の成分が全て 0 である)．

$$\mu\mathbf{I}-B=LU$$

$$=\begin{pmatrix} 1 & 0 & 0 & & 0 \\ m_1 & 1 & 0 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & m_{n-2} & 1 & 0 \\ 0 & & & m_{n-1} & 1 \end{pmatrix}\times\begin{pmatrix} \alpha_1 & \beta_1 & 0 & & 0 \\ 0 & \alpha_2 & 0 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ & & 0 & \alpha_{n-1} & \beta_{n-1} \\ 0 & & & 0 & \alpha_n \end{pmatrix}$$

(21) の解は

$$\begin{aligned} L\mathbf{y} &= \mathbf{x}_k \\ U\mathbf{x}_{k+1} &= \mathbf{y} \end{aligned} \tag{22}$$

を解くことにより得られる．証明は省略するが,(22) は次のように容易に解を求めることができる．

i = 2 〜 n に対して　　$x_i=x_i-x_{i-1}\times m_{i-1}$

$x_n=\dfrac{x_n}{\alpha_n}$

i = n-1 〜 1 に対して　　$x_i=(x_i-\beta_i\times x_{i+1})/\alpha_i$

2.4　逆変換

ハウスホルダ変換の第 k 段階における列ベクトル $\mathbf{w}$ 及び行列 Q を $\mathbf{w}_k, Q_k$ と表せば

$$\begin{aligned} B &= Q_{n-2}^{-1}(...(Q_2^{-1}(Q_1^{-1}AQ_1)Q_2)...)Q_{n-2} \\ &= (Q_{n-2}^{-1}...Q_2^{-1}Q_1^{-1})A(Q_1Q_2...Q_{n-2}) \qquad (23) \\ & \qquad (24) \end{aligned}$$

従って，A の固有ベクトル $\mathbf{v}_A$ と，A と相似な三重対角行列 B の固有ベクトル $\mathbf{v}_B$ との間に

$$\begin{aligned} \mathbf{v}_A &= Q_1 \times Q_2 \times ... \times Q_{n-3} \times Q_{n-2} \times \mathbf{v}_B \\ &= (\mathbf{I} - c_1\mathbf{w}_1\mathbf{w}_1^T) \times (\mathbf{I} - c_2\mathbf{w}_2\mathbf{w}_2^T) \times \qquad (25) \\ & \quad ... \times (\mathbf{I} - c_{n-3}\mathbf{w}_{n-3}\mathbf{w}_{n-3}^T) \times (\mathbf{I} - c_{n-2}\mathbf{w}_{n-2}\mathbf{w}_{n-2}^T) \times \mathbf{v}_B \qquad (26) \end{aligned}$$

なる関係がある．この関係を用いて，逆反復法で求めた三重対角行列の固有ベクトルから元の行列 A の固有ベクトルを計算するのが逆変換である．

ハウスホルダ変換の第 k 段階において，$\mathbf{w}_k$ を計算し終った時は，A の k 番目の列が不要になるから，$\mathbf{w}_k$ をここに書き込むとメモリを節約できる．

3　ハードウェア記述言語 VHDL

3.1　基本概念

ハードウェア記述言語は並行的に動作するハードウェアを記述するために開発された言語である．従って，各命令が順番に実行されていく普通のソフトウェアプログラミング言語にはない, 幾つかの特有な概念を持っている．ここでは簡単な例を通して VHDL の特徴となる基本概念を説明する．

以下は排他的論理和回路 (exclusive or) を VHDL で記述した例である

```
library ieee;
use ieee.std_logic_1164.all;

entity MyXor is
  port(
        x : in std_logic;
        y : in std_logic;
        z : out std_logic
      );
end MyXor;

architecture Behavior of MyXor is
begin
        z <= x xor y; - - exclusive or
end Behavior;
```

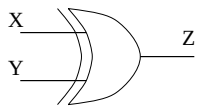


図 3: 排他的論理和回路

このように， VHDL ソースは一般的に, **library** 等で始まるパッケージ宣言， **entity** で始まるエンティティと **architecture** で始まるアーキテクチャの 3 つの部分から構成されている．エンティティの中にある x, yとz はこの回路の入出力を表すポートである．

```
z <= x xor y;
```

により z の値が x と y の値によって決まる．各ポート (一般的に信号 (後述)) の値はこのように信号代入文と呼ばれるステートメントによって関連づけられる．また, 上記の std_logic は一種のデータ型である．

以下に順にこれらの基本概念について説明していく．

3.1.1　パッケージ宣言

VHDL の中で頻繁に使われるデータ型や演算機能等をまとめたものがパッケージである．パッケージの集まりがライブラリである．

library ライブラリ名;

で，どのライブラリを使うか指定する．1 つのライブラリが多くのパッケージを含むから，どのパッケージを使うかは

use ライブラリ名. パッケージ名. **all** ;

で指定する．これらのパッケージ宣言は普通，ソースの一番上に記述する．

上の **all** は予約語で，パッケージの全てを使用するのを意味する．パッケージの一部だけを使いたい場合は **all** の変わりにその部分の名前を記述すればよい．3.1の例では，ieee という標準ライブラリと，このライブラリの中の std_logic_1164 というパッケージを宣言している．この宣言により std_logic 等の基本的なデータ型が使えるようになる．

3.1.2　エンティティ

これは設計しようとしている回路の入力と出力を定義する部分である．回路を 1 個のチップに例えれば，エンティティがチップのピン形状やピン配置などを規定する仕様書に相当する．基板を設計する時に, 使用するチップのピン配置情報が必要になるのと同様に，エンティティの中に書かれている情報も，設計している回路が他の回路に組み込まれる時に必要になる．

エンティティの一番簡単な構文[6]は次の通りである．

[6] ここではポートのみを宣言しているが，ポート宣言の他に **generic** 宣言と **signal** 宣言がある．

```
entity エンティティ名 is
    port (
      ポート名 : モード データ型;
      ポート名 : モード データ型;
      ....
      ポート名 : モード データ型
   );
end エンティティ名;
```

エンティティ名はエンティティを識別するための文字列である．VHDL で使う識別子には幾つかの規定 (詳しくはマニュアルを参照) があるが，アルファベット文字と数字の組み合わせで，数字から始まらない文字列を使えばまず問題はない．ポートは回路の入出力端子である．ポート名はエンティティ名と同じく，識別子である．モードはポートを流れる信号の方向を決めるパラメータで， **in** ， **out** , **inout** (それぞれ入力，出力，双方向) 等の値を取ることができる．データ型は対応する入出力信号のタイプである．ポート宣言文はセミコロン (;) で区切り，最後の文にはセミコロンがない．

前例では，エンティティ名が myxor で， x と y という 2 つの入力端子 (**in**) と z という 1 つの出力端子 (**out**) を持つ回路を宣言している．これらの端子を流れる信号の型はいずれも std_logic(ディジタル信号を表す最も基本的なデータ型) である．

3.1.3　アーキテクチャ

エンティティが回路と外部との接続を定義しているのに対して，アーキテクチャは回路の内部的な機能を記述する部分である．エンティティをチップのピン配置を示す仕様書に例えれば，アーキテクチャがチップ内部の回路図に相当する部分である．アーキテクチャの構文は一般的に次の通りである．

Architecture アーキテクチャ名 **of** エンティティ名 **is**
　アーキテクチャ宣言部
begin
　アーキテクチャ本体
end アーキテクチャ名;

アーキテクチャ宣言部には，アーキテクチャ本体の中で使われるローカルな信号や定数等を宣言する．3.1の例ではこの部分はない．

アーキテクチャ本体の中に，アーキテクチャ宣言，信号代入文，プロセス文などを書くことにより，設計している回路の機能を記述する．3.1の例は，一つの信号代入文で記述されている．

```
z <= x xor y - - exclusive or
```

VHDL では 1 つの回路は 1 つのエンティティしか持てないが，アーキテクチャは何個でも持つことができる (同じ機能を色々な方法で実現できるのと同じである)．アーキテクチャ名 (前例のアーキテクチャ名は Behavior である) はアーキテクチャ同士を識別するための名前である．アーキテクチャが 1 つしかない場合は自動的に **Architecture** 文で指定したエンティティに結ばれるが，2 つ以上ある場合は，どのアーキテクチャを使うか，**configuration** 文で明確に指定しなければならない．また，**PeakVHDL&FPGA** ではエンティティとアーキテクチャは同じファイルの中に記述しなければならない．

3.1.4　データ型・信号・信号代入文

VHDL の信号は実際の回路の配線に相当する．信号代入文により信号に値を与える．信号に代入するすることのできる値は信号の宣言文で宣言したデータ型によって決まる．ソースレベルでは色々なデータ型があるが，論理合成 (ゲートレベルの実際のディジタル回路に変換) すると全てのデータ型が H かまたは L の値を持つ信号の集まりとして表現される．例えば，大きさが 0 〜 127 の整数型の信号を定義すると，実際の回路では 8 本の配線になる．信号代入文の左辺には値を代入したい信号を，右辺には結果的に左辺と同じデータ型を持つ値として評価される式を書く．右辺と左辺は記号 <= で結ぶ．前の例では，x と y と z は std_logic というデータ

型を持つ信号 (ポートは信号として見なされる) である． std_logic は標準パッケージ std_logic_1664 の中で定義されているデータ型で以下の値を取れる (値が ’ で囲まれていることに注意).

表 1: std_logic データ型

値	意味	値	意味	値	意味
'U'	初期化されていない値	'L'	弱い信号の 0	'0'	0
'X'	不定	'H'	弱い信号の 1	'1'	1
'W'	弱い信号の不定	'-'	don't care		

このデータ型には表 2のような論理演算が予め定義されている

表 2: std_logic の論理演算

演算記号	意味
not	否定
and	論理積
or	論理和
xor	排他的論理和

前記の例は

```
z <= x xor y ;  - - exclusive OR
```

の信号代入文で z に x と y の排他的論理和を代入しているのである． - - から行末までの部分はコメントで，コンパイラによって無視される． **xor** を他の演算記号 (例えば **and** と **or**) に置き換えれば論理和や論理積等も簡単に作れる.

```
z <= x or y;
```

```
z <= x and y;
```

上の代入文では，右辺の評価値が変化するとその変化が即座に左辺に伝わるが，実際の回路のように遅延を持たせたい場合は次のように代入文右辺の後にキーワード **after** と遅延時間を追加すれば良い.

```
z <= x xor y after 10 ns ;
```

このように書くと， 10ns の遅延時間を持つ回路を実現できる．図 4にはこの回路のシミュレーション結果が示されている．この図では 0 〜 10ns までの間に, 値 (x **xor** y) が信号 z に伝わっていないため， z が'U'(初期化されていない値) を取る.

図 4: 遅延信号代入文のシミュレーション結果 [7]

3.1.5　条件つき信号代入文

今までの信号代入文は，いずれも右辺の値が無条件で左辺に代入されていたが，ある特定の条件が成り立った時にのみ代入を実行するように，記述することができる．

条件付き信号代入文

信号 <= 式 1　**when**　条件 1　**else**
　　式 2　**when**　条件 2　**else**
　　...
　　式 n　**when**　条件 n　**else**
　　式 $n+1$;

コンパイラはまず条件 1 を評価する．条件 1 が成り立っていれば式 1 が代入される．そうでなければ，条件 2 を評価し，条件 2 が成り立っていれば式 2 が代入される．このように順番に条件を評価していき，どの条件も成立していなければ最後の式 $n+1$ が代入される．

前述した例では，VHDL で予め定義されている論理演算 **xor** を使っていたが，条件つき信号代入文により次のように書くこともできる．[8]

```
z <= '0' when   x='0' and y='0' else
     '1' when   x='0' and y='1' else
     '1' when   x='1' and y='0' else
     '0' ;
```

3.1.6　配列

データバス等のように，同じ機能を持った信号で構成されている信号グループは，VHDL では配列として記述する．任意のデータ型の配列を **type** 文により新しく定義することができるが，std_logic の配列は予め std_logic_vector として定義されて

[7] ファイル名 = ./image/sim-re/sr-delay.ps

[8] 厳密に言うと，この記述は **xor** と全く同じではない．std_logic 型は '0' と '1' の他に 'U' や'X' 等の値も取れるから，'X' **xor** 'X' は 'X' として定義されているあるが，上の例では条件つき信号代入文の最後の **else** の後にある '0' になる．しかし，'X' はシミュレーションを行なうための抽象的な値で実際には存在しないので，論理合成すれば **xor** を使った設計も **when** を使った設計も同じ回路に変換される．

いる．

配列型の信号は普通の信号と同様に宣言できる．但し，データ型のところでは次のように書く

配列名 (最小のインデックス　**to**　最大のインデックス) または
配列名 (最大のインデックス　**downto**　最小のインデックス)

例えば，4，8，12 ビットの双方向データバスは次のように宣言する．

data4 :　**inout**　std_logic_vector(0 to 3);
data8 :　**inout**　std_logic_vector(1 to 8);
data12 :　**inout**　std_logic_vector(15 to 4);

このように配列のインデックス範囲は，任意の値から任意の値まで，昇順または降順で宣言できる．

配列の使用例として，4 ビットマルチプレクサを挙げる (図 5)．このマルチプレクサは，sel の値に応じて 4 つの入力 $x(0), x(1), x(2)$と$x(3)$ の内，1 つを選択して，出力 y に接続する回路 (表 3と図 6を参照) である．

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX is
    port (
        sel: in std_logic_vector(1 downto 0);
        x: in std_logic_vector(3 downto 0);
        y: out std_logic
    );
end MUX;

architecture BEHAVIOR of MUX is
begin
    y <= x(0) when sel="00" else
         x(1) when sel="01" else
         x(2) when sel="10" else
         x(3);
end BEHAVIOR;
```

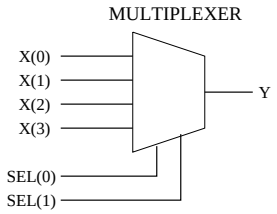


図 5: マルチプレクサ回路

表 3: マルチプレクサの真理値表

sel の値	y に接続される入力
“00”	$x(0)$
“01”	$x(1)$
“10”	$x(2)$
“11”	$x(3)$

上のマルチプレクサ回路の例では 2 番目の方法 (降順の配列, つまり sel=”10” なら $sel(1)$=’1’,$sel(0)$=’0’) により配列型の信号を 2 つ宣言している． sel は 2 ビットからなる信号で， x は 4 ビットからなる信号である．また，ソースの中に書かれているように，配列の 1 部を参照するには表 4のように配列名の後に，参照したい部分のインデックスを丸括弧で囲んで記述する．

表 4: 配列の参照

記述	参照される対象
$x(0)$	最も右にあるビット
$x(3)$	左から見て 1 番目のビット
$x(2$ **downto** $0)$	x(2),x(1),x(0)

このマルチプレクサを **PeakVHDL&FPGA** でシミュレートする場合に使うテストベンチとシミュレーション結果[9]を参考までに以下に示す．

```
library ieee;
use ieee.std_logic_1164.all;

use std.textio.all;
use work.MUX2;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
  component MUX2 is
    port (
       sel: in std_logic_vector(1 downto 0);
       x: in std_logic_vector(3 downto 0);
       y: out std_logic  );
  end component;

  constant PERIOD: time := 100 ns;
  -- Top level signals go here...
```

[9]シミュレーション方法とテストベンチの作成については第 5章を参照.

```
  signal sel: std_logic_vector(1 downto 0);
  signal x: std_logic_vector(3 downto 0);
  signal y: std_logic;
begin
  DUT: MUX2 port map ( sel,x,y);

  sim : process  begin
     x <= "0101";
     sel <= "00"; wait for PERIOD;
     sel <= "01"; wait for PERIOD;
     sel <= "10"; wait for PERIOD;
     sel <= "11"; wait for PERIOD;
     x <= "1010";
     sel <= "00"; wait for PERIOD;
     sel <= "01"; wait for PERIOD;
     sel <= "10"; wait for PERIOD;
     sel <= "11"; wait for PERIOD;
     wait;
  end process sim;

end stimulus;
```

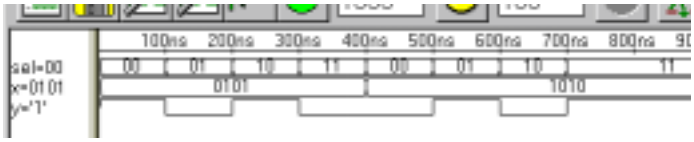


図 6: マルチプレクサのシミュレーション結果 [10]

3.1.7　列挙型

順序回路の状態を表すために，従来，0 と 1 の組み合わせによる方法がよく使われていた．例えば，[スタンドバイ]，[動作] と [停止] という 3 つの状態を持った回路を 2 ビットで次のように表せる．

表 5: ビットによる表現

回路の状態	ビット値
スタンドバイ	0 0
動作	01
停止	11

[10] ファイル名 = ./image/sim-re/sr-mux.ps

回路の状態が少ない時は，このような対応関係はそれほど問題にはならないが，状態数が数十あるいは数百にも及ぶ大規模なディジタルシステムになると，紛らわしく，どの値がどの状態を表しているのか把握できなくなる．

VHDL では， **type** 文により，このような有限個数の値を取るデータを表現するための新しいデータ型を簡単に定義できる．

type 新しいデータ型 **is** (値 1，値 2，．．．);

例えば，上の回路状態は次のように定義する．
type State_type **is** (Standby,Running,Stop);
今まで使ってきた std_logic 型も実はパッケージ std_logic_1164 の中で列挙型として定義されている．

```
........ 省略
TYPE std_ulogic IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
........ 省略
SUBTYPE std_logic IS resolved std_ulogic;
........ 省略
```

3.2　同時処理文と順序処理文

ハードウェア記述言語と, 他のソフトウェア的なプログラミング言語との一番大きな違いはハードウェア記述言語が同時に処理される命令を持っている点にある．

アーキテクチャの **Begin** と **End** の間にある命令文は同時に実行される同時処理文である．これらの命令文はどの順番で記述しても実行結果が変わらない．従って,a, b, c, d を同じデータ型を持った信号とすれば，次の 2 つのアーキテクチャは全く同様の動きをする．

```
a <= b;
b <= c;
c <= d;
```

```
b <= c;
c <= d;
a <= b;
```

今まで使ってきた信号代入文はみな式により左辺の信号に値を代入するため，簡単な組み合わせ回路しか作れない．プロセス文では， C や Pascal 等ソフトウェア的なプログラミング言語の制御構造も使うことができるので，非常に複雑な回路も簡潔かつ分かりやすく記述できる．プロセス文は一般的に次のように書く．

プロセス名： **process**（信号 1，信号 2，．．．）
　変数宣言
begin
　順序処理文
end process プロセス名;

信号 1，信号 2，．．．はプロセスのセンシティブティ・リストと呼ばれ，ここに書かれている信号の 1 つでも変化するとプロセス文が実行される． **begin** から **end process** までの部分は順序処理文で，ここに書かれる命令は順番に上から下へ実行されていく．順序処理文には選択構造を表す **if** 文や反復構造を表す **loop** 文などがある．

3.2.1　if 文

if 文は次の二形式の内のいずれかで記述する．

if 条件 **then**
　順序処理文
endif

if 条件 **then**
　順序処理文
else
　条件処理文
endif

if 条件 **then**
　順序処理文
elsif 条件
　順序処理文
elsif 条件
　順序処理文
　.....
else
　条件処理文
endif

上の 順序処理文 の中にまた **if** 文と他順次処理文を書くことができる．前の排他的論理和 myxor は **if** 文で次のように書き直せる．

```
..... 省略
if (x='0') then
  if (y='0) then
     z <= '0';
  else
     z <= '1';
  end if
else
  if (y='0) then
     z <= '1';
  else
     z <= '0';
  end if
end if;
..... 省略
```

```
..... 省略
if (x='0' and y='0') then
     z <= '0';
elsif (x='0' and y='1') then
     z <= '1';
elsif (x='1' and y='0') then
     z <= '1';
else
     z <= '0';
end if;
..... 省略
```

else のない **if** 文を用いる時，1 つ注意しなければならないことがある．**if** 文の条件項にすべての場合を記述しておく必要がある．

例えば，右上のリストの最後の **else** は **elsif** (x='1' **and** y='1') **then** z <= '0'; と同等であるように見えるが，もしここで **else** の変わりにこのように **elsif** を使うと，x='X' の時にどの信号代入文も実行されないため，z の値が保持される，従って，ラッチ回路が生成されることになる．もちろん意図的にラッチ回路を作る場合はこのように記述するのだが，組み合わせ回路を生成しようとして，誤ってラッチ回路を生成してしまうようなこともある．

3.2.2　case 文

if 文と同様，選択構造を表す命令文である．

case 変数 (または信号) **is**
　when 値 1 => 順序処理文 1
　when 値 2 => 順序処理文 2
　.....
　when others => 順序処理文 n
end case ;

case の後に指定される変数 (信号) が値 1 を取る時は順序処理文 1 が実行され，値 2 を取る時は順序処理文 2 が処理される．残りの値は **others** によってカバーされる．**case** を用いた排他的論理和の記述例を以下に示す．

```
library IEEE;
use IEEE.std_logic_1164.all;

entity myxor is
   port( x : in std_ulogic;
         y : in std_ulogic;
         z : out std_ulogic
        );
end myxor;

architecture RTL of myxor is
begin
   process(x,y)
      variable tmp : std_logic_vector(0 to 1);
   begin
      tmp := x & y;
      case tmp is
          when "00"   =>  z <= '0';
          when "01"   =>  z <= '1';
          when "10"   =>  z <= '1';
          when others =>  z <= '0';
          end case;
    end process;
end;
```

上の tmp は変数と呼ばれる．変数はプロセスの **begin** の前に

variable 変数名 : データ型;

で宣言する．値を保持するという点では変数も信号と同じである．信号に使えるデータ型は変数にも使える．しかし，変数はそれが宣言されているプロセスの中でしか使うことができない．従って，変数により他のプロセスと通信することはできない．変数と信号の違いについては後述するが，ここでは， VHDL の変数は基本的に C や Pascal の変数と同じであると考えて良い．

& は結合演算子と呼ばれ，配列または何個かのスカラーをまとめて新しい配列を生成するものである．上の例では, tmp が昇順配列として宣言され,

```
tmp:= x & y;
```

により， $tmp(0)$ に x を， $tmp(1)$ に y を代入する．このように，信号の代入には <= を，変数の代入には := を使う．

3.2.3　loop 文

loop 文は，同じ構造を何個も含んでいる回路を簡潔に記述するために使う．以下はその書式である．

for ループ変数 **in** 下限 **to** 上限 **loop**
　順序処理文
end loop ;

文法的には，ループ変数の下限と上限は動的 (実行して，始めて値が定まり，実行中でも変化できる) であっても間違いではないが，論理合成する場合は，普通ループの一回の反復当たりに 1 つの回路が生成されるから，動的な値をここに使うと論理合成できなくなる．この点は，ハードウェア記述言語とソフトウェア的なプログラミング言語との大きな相違の 1 つである．多くの計算において，動的な反復構造 (例えば $n \times n$ 行列の成分を読み込む時，n が変化する場合) は不可欠であるが，第 6 章でその実現方法を述べる．

3.2.4　wait 文

wait は以下のいずれかの書式で使う．

1. **wait on** 信号 1，信号 2，．．．；
2. **wait until** 条件;
3. **wait for** 時間;
4. **wait** ;

wait 文はプロセスの実行を一時停止する．(1) の場合は，信号 1，信号 2，．．．の内，1 つでも変化すると実行が再び再開される．(2) の場合は，指定した条件が成立するまで実行が停止される．(3) の場合はプロセスが指定した時間だけ停止される．(4) のように何も指定しない場合は実行が永遠に止まる．

(3) と (4) はテストベンチの中で，テストデータを生成するためによく使われる．例えば，**wait** で次のように，簡単に周期 20 ns の方形波を作ることができる．

```
... 省略 ....
architecture GenerateClock of Test is
    signal clock = '0';
```

```
begin
     wait for 10 ns;
     clock <= not clock;
end GenerateClock;
```

3.3　解決関数とデルタ遅延

同時処理文は，それぞれが他の同時処理文と関係なく独立に動作するため，1 つのアーキテクチャの中に複数の同時処理文を記述した場合，次のような問題が発生する．

1. 同じ信号に，2 つ以上の同時処理文により異なった値を代入した場合，結果としてどのような値がその信号に代入されるのか？
2. ある同時処理文により，ある信号 A の値が変化した場合，ちょうどその信号 A が変化した時刻に，別の同時処理文が A を入力として読むと，A が変化する直前の値，それとも変化した直後の値が入力されるのか？

以上の問題を解決するために，VHDL では解決関数とデルタ遅延という 2 つ特別な概念を導入している．

3.3.1　解決関数・解決データ型

同時信号処理文で信号に値を代入するとその信号に対して 1 つのドライバが生成される．プロセス文の中で，順序処理文により信号に値を何回も代入することができるが，各代入が順番に実行されるため，複数回の代入があっても結果的に 1 つのドライブしか生成されない．1 つの同時処理文により，同じ信号に対しては 2 つ以上のドライブを生成することができないのである．同じ信号に対してドライバが 2 つ以上 (別々の同時処理文により) 生成された場合，その信号のデータ型に対して解決関数を定義しておく必要がある．解決関数 (resolution function) は，同じ信号に，複数のドライバにより同時に値が代入される場合，信号の最終的な値を決定する関数である (ドライバの衝突を解決 =resolve する意味から resolution という)．解決関数が定義されているデータ型は解決データ型 (resolved type) と呼ばれる．例えば，

std_logic はパッケージ std_logic_1164 の中で，次のように解決型として定義されている．

```
........ 省略
TYPE std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H','-');
FUNCTION resolved (s : std_ulogic_vector) RETURN std_ulogic;
SUBTYPE std_logic IS resolved std_logic;
........ 省略
```

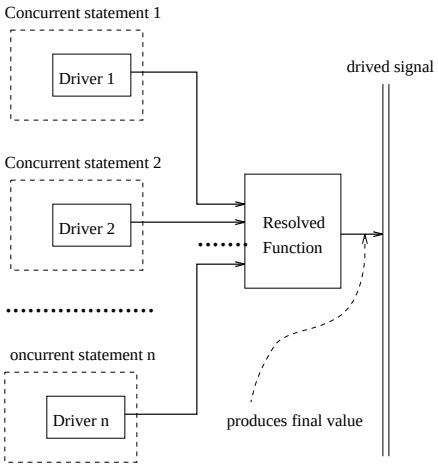


図 7: 解決関数の概念図 [11]

上の resolved は解決関数で，各ドライバの出力がこの関数 resolved の入力となる (複数の入力があるから，入力が配列として宣言されている)．

例として以下のようなソースを考える．左のソースは，解決データ型でない std_ulogic に 2 つの同時処理文により異なる値を代入するから，コンパイルするとエラーになる [12]．それに対して，右のソースは解決データ型である std_logic を用いているから，エラーは生じない[13]．また'1' と'0' をパラメータとして解決関数 resolved に与えると返し値'X' が得られるので，シミュレートする時， x に 'X' が入る[14]．

---

[11] ファイル名 = ./fig/resfunc.xfig.eps

[12] **PeakVHDL&FPGA** のコンパイラはこのエラーを発見できない．おそらくプログラムのバグであろう．

[13] **MAX+plus II** は複数ドライバをサポートしないため，このソースは **MAX+plus II** ではコンパイルできない． **PeakVHDL&FPGA** の論理合成ツールも複数ドライバをサポートしないから，このソースを **PeakVHDL&FPGA** で論理合成することはできない．

[14] **PeakVHDL&FPGA** でシミュレートすると最後の代入文 (ここでは '1') が x の最終的な値になる．これもバグであると思われる．

```
library ieee;
use ieee.std_logic_1164.all;
entity UsingUnresolved is
   port (
      x: out std_ulogic
      );

end EX1;
architecture BEHAVIOR of UsingUnresolved is
begin
     x <= '0';
     x <= '1';
end BEHAVIOR;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity UsingResoleved is
   port (
      x: out std_logic
      );

end EX1;
architecture BEHAVIOR of UsingResolved is
begin
     x <= '0';
     x <= '1';
end BEHAVIOR;
```

3.3.2　デルタ遅延

VHDL は逐次型 (von Neumann 型) コンピュータで処理されるため，各命令を同時に実行することはできない．従って，同時処理文

信号 <= 式

を処理する時，右辺の式を計算して，すぐに左辺の信号に代入すると処理の順番によって結果が一意に定まらないことがある．

図 8にあるような回路を考える． D フリップフロップはクロック信号 CLK の立ち下がりで信号 D を Q に出力する．クロックが立ち下がらない限り， Q が同じ値に保持され続けるから， D が変化してもその変化が Q には伝わらない．

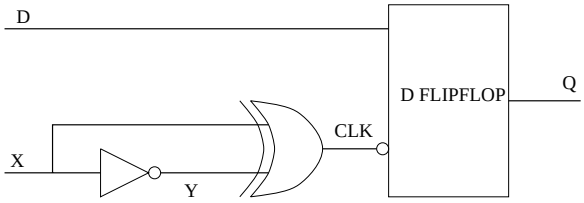


図 8: デルタ遅延の説明図 [15]

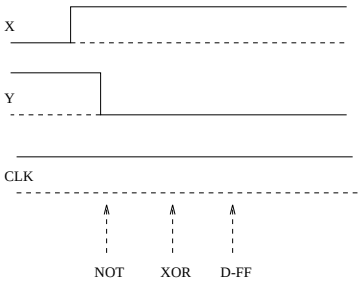


図 9: フリップフロップが動作しない

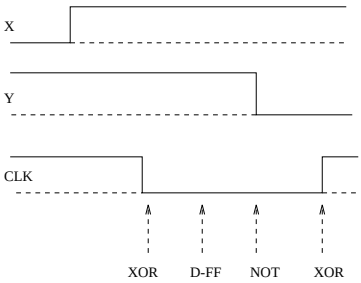


図 10: フリップフロップが動作する

この回路をシミュレートする時，回路の各部 (not 回路， xor 回路と D フリップフロップ) を順番に処理していかなければならない．例えば， X が 0 から 1 に変わる場合を考える． not 回路 xor 回路 D フリップフロップ の順番で実行するとクロックの立ち下がりが検出されないので Q の値が保持される．しかし， xor 回路 D フリップフロップ not 回路 の順番で実行するとクロックの立ち下がりが検出され， Q の値が保持されないという違った結果が得られる．

[15] ファイル名 = ./fig/delta.xfig.eps

このように，代入される値の計算と代入を同時に実行すると明らかにシミュレーション結果が一意に定まらない．VHDL では，同じ回路をどのシミュレータを使っても同じシミュレーション結果が得られるように，代入される値の計算と代入が別々の時刻で行なわれる．まず，信号代入文の右辺の式をすべて計算して，代入される値を求める．必要な計算が全て終ってから始めて実際の代入を実行する．つまり，ある信号が変化した場合，ちょうど信号が変化する時刻に，あるプロセスがその信号を入力として読むと，変化直前の値を読むことになる．同じ図 8の回路をこの方法でシミュレートすると，図 11と図 12に示されているように，どの順番で処理しても最終的な結果が同じである．

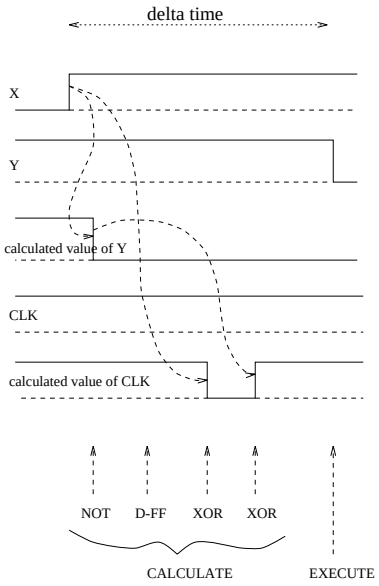


図 11: not 回路を先に処理する場合

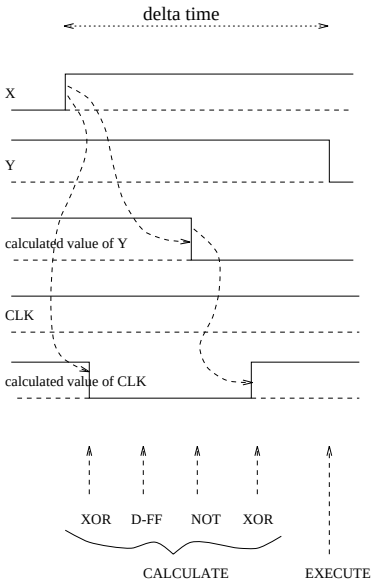


図 12: xor 回路を先に処理する場合

計算と代入を別別の時刻で行なうことは，入力が変化してから，その変化が出力に反映されるまでの間に一種の遅延が存在することを意味する．この遅延がデルタ遅延と呼ばれる．デルタ遅延は，シミュレーション結果を一意に定めるためのもので，実際のシミュレーション結果には現れることがなく，シミュレータの内部でしか見ることができない．

4 MAX+plus II の使用方法

MAX+plus II は Altera 社の FGGA をターゲットとしたディジタル回路の設計を支援するソフトウェアである．**MAX+plus II** は主に以下のような機能を持っている．

- ハードウェア記述言語のコンパイラ

  MAX+plus II は元々 VerilogHDL を対象とした EDA ツールであるが，最近 VHDL にも対応できるように機能追加された．しかし VerilogHDL コンパイラに比べ，VHDL コンパイラはバグが多く，意味不明なメッセージをよく出力する．VHDL は他ベンダーのコンパイラを使用することが推奨されている．

- 時間遅延を考慮したシミュレータ

- 配置配線ツール

  ディジタル回路を表す EDIF ファイルから Altera 社の FPGA 用書き込み情報を生成する．業界標準フォーマットの EDIF ファイルを入力とするため，他ベンダーのツール (例えば後述の **PeakVHDL&FPGA**) とのインタフェースは容易にとれる．

- 回路図入力ツール

  マウスで回路図をグラフィカルに入力できる．入力した回路図は VerilogHDL または VHDL ソースに自動的に変換できる．

MAX+plus II には UNIX 版と Windows 版がある．当研究室にある Windows 版の **MAX+plus II** は一部のデバイスにしか対応しない．他に，コンパイラにも色々な制限がかけられており，部分的に使用できないメニューがある．UNIX 版の **MAX+plus II** はフルバージョンで Altera 社の全てのデバイス (もちろんソフトウェアが出荷される前のデバイス) に対応し，全機能が使える．

本章は VHDL ソースの作成からシミュレーション, 論理合成までの一連の作業を通して **MAX+plus II** の使い方を説明する．

4.1 環境設定

UNIX 版の **MAX+plus II** は使う前に若干の環境設定が必要である．以下の説明では，**MAX+plus II** が /local2/maxplus2 にインストールされた場合であるが，他のディレクトリにインストールされた場合は /local2/maxplus2 を実際のディレクトリに置き換えれば良い．
まず，自分のホームディレクトリにある.cshrc ファイルに以下のコマンドを追加する．

```
set path=(\$path /local2/maxplus2/bin)
setenv LM_LICENSE_FILE /local2/maxplus2/adm/license.dat
```

次に，インストールディレクトリにある maxplus2.ini ファイルををホームディレクトリにコピーする．

```
%cp /local2/maxplus2/maxplus2.ini ~/
```

これで，環境設定が完了する．

1 つの VHDL ソースをコンパイルしたり，シミュレーションを行なったりすると関連するファイルがたくさん作られる．**MAX+plus II** は，これらのファイルを 1 つのプロジェクトとして管理する．プロジェクトの名前はトップレベルのモジュールの名前と同じである．またプロジェクトの中で設定した情報はプロジェクトと同じ名前で，拡張子が acf のファイルに保存される．

MAX+plus II によって作られたファイルを間違えて消すと，最初からやり直さない限り，**MAX+plus II** が動作しなくなる場合が多いから，一つのプロジェクト (一つの回路設計) に対して，予めディレクトリを作り，関連ファイルを全てそのディレクトリに保存するようにした方が良い．

4.2 MAX+plus II の起動

以下のコマンドで **MAX+plus II** を立ち上げる．

```
%maxplus2
```

標準設定では,UNIX 版の **MAX+plus II** の外観と操作方法は Windows95 版のものとほんとんど同じである．メニューの選択やスクロールバー等は左ボタンで，ショー

トカットメニューのポップアップ等は右ボタンで行なう.

MAX+plus II の操作は基本的に一番上にあるメニューバーまたはメニューバーの下にあるツールバー (図 13) により操作する.

図 13: **MAX+plus II** のメニューバーとツールバー [16]

メニューバーもツールバーも現在アクティブになっている子ウィンドウにより変わるが, どれも以下の共通項目を含んでいる.

- **MAX+plus II**
  MAX+plus II はコンパイラー, テキストエディタ, シミュレータ等の幾つかのアプリケーションから構成されている. このメニューでこれらのアプリケーションを起動する.

- File
  新しいファイルを作成したり, ファイルを保存したりする.

- Assign
  使用するデバイスを指定したり, デバイスのピンを割り当てたりする.

- Options
  表示色やなどの外観的な設定をする

- Help
  MAX+plus II に関する詳細な説明を見る.

4.3 新しいプロジェクトの作成

新しい設計はプロジェクトの作成から始まる. プロジェクトを新規作成するには以下の操作を行なう.

[16] ファイル名 = ./image/max/maxplus2.ps

1. メニューバーの File をクリックして， File メニューを開く．

2. File メニューの Project をクリックして，サブメニューを開く．

3. サブメニューの Name ... を選択する．[17]

画面に，プロジェクト名を入力するためのダイアログ (図 14) が表示される．普通カレントディレクトリにあるファイルが全部左側にリストされるが， *Show Only Tops Of Hierachies* をチェックすればプロジェクトのみが表示され，見やすくなる．

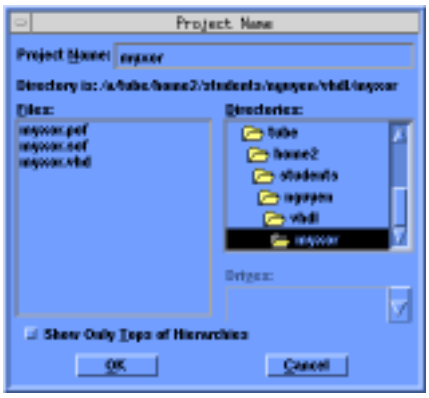


図 14: プロジェクトの新規作成 [18]

プロジェクト名 (ここでは myxor) を入力してから OK ボタンをクリックすると新しいプロジェクトが作られる．

4.4 VHDL ソースの作成

File ⇒ New ... を選択する．

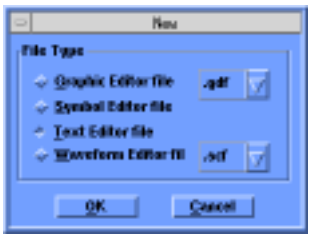


図 15: ファイル種類の選択 [19]

[17] 簡単のため，以降このような操作を単に File ⇒ Project ⇒ Name ... と表記する．

[18] ファイル名 = ./image/max/dlg-proj.ps

[19] ファイル名 = ./image/max/new.ps

出てくるダイアログの *Text Editor file* をチェックして，OK ボタンをクリックすると，テキストを入力するためのウィンドウが表示される．以下の内容を入力する．

```
library ieee;
use ieee.std_logic_1164.all;

entity MyXor is
  port(
        x : in std_logic;
        y : in std_logic;
        z : out std_logic
      );
end MyXor;

architecture Behavior of MyXor is
begin
        z <= x xor y;
end Behavior;
```

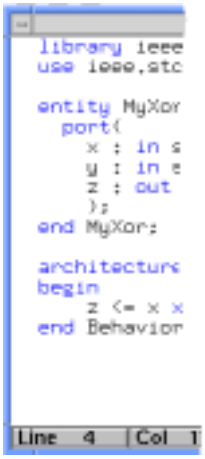


図 16: テキストエディタウィンドウ

これは排他的論理和回路を VHDL で記述したものであるが，VHDL の詳細については第 3章の **ハードウェア記述言語 VHDL** を参照されたい．

入力が終ったら，File ⇒ Save で作成したソースを myxor.vhd という名前のファイルとして保存する．

4.5　VHDL ソースのコンパイル

MAX+plus II では，VHDL などのソースをコンパイルすることにより，ソースファイルを解析してシミュレーションするために必要な情報と実際のデバイスに書き込むための情報等を得る．コンパイラは，VHDL ソースだけでなく，AHDL と他の処理系で作ったネットリストファイルも入力として使える．また，Altera のデバイスに書き込むための情報だけでなく，他の処理系でもシミュレートできる，タイミング情報を含んだ VHDL ソースも出力することができる．

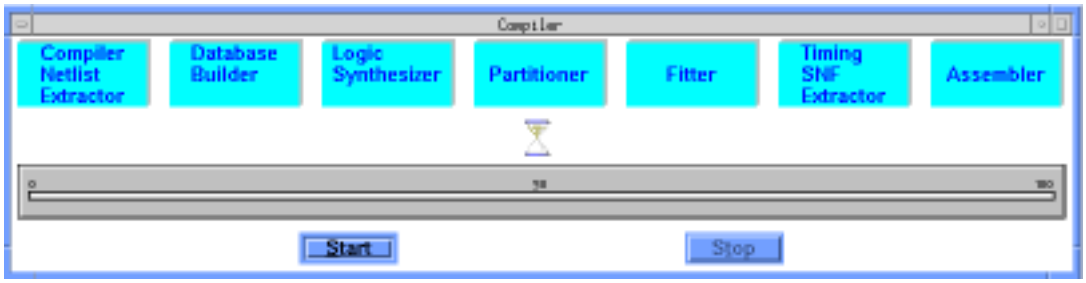


図 17: コンパイラウィンドウ [20]

MAX+plus II ⇒ Compiler でコンパイラウィンドウを開いて，Start ボタンをクリックすれば先ほど作った myxor.vhd がコンパイルされるが，**MAX+plus II** が常にデバイスに書き込むことを前提とするので，コンパイルする前にデバイスを Assign ⇒ Device ... で指定しなければならない．

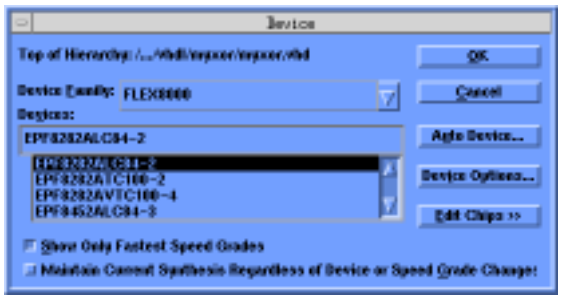


図 18: デバイスの指定 [21]

例えば，CQ 出版社の基板に乗っている EPF8282ACL84-2 を使う場合は，*Device Family* に FLEX8000, *Devices* に EPF8282ACL84-2 を指定する．**MAX+plus II** に自動的にデバイスを決定させたい場合は *Devices* に *AUTO* を指定すれば良い．

ソースファイルにエラーがあった場合は，コンパイラが止まり，エラーメッセージを表示する (図 19)．エラーメッセージをダブルクリックするか，またはエラーメッセージをクリックしてから Locate ボタンをクリックするとカーソルがエラーの発生した箇所に移動する．Help on Message をクリックすれば，発生したエラーの原因と対策についての説明を見ることができる．エラーがなく，コンパイルが正常に終了すると，幾つかのファイルが新しく作られる．

[20] ファイル名 = ./image/max/comp.ps

[21] ファイル名 = ./image/max/device.ps

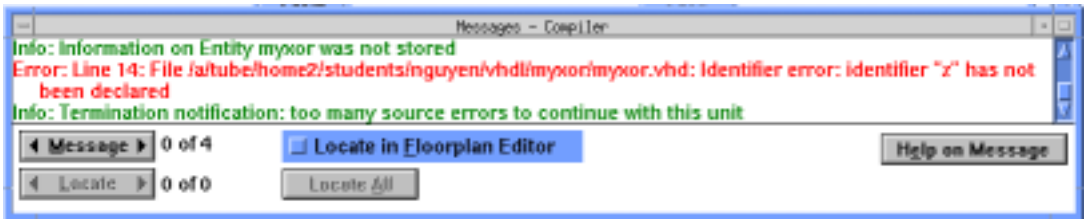


図 19: メッセージウィンドウ [22]

MAX+plus II ⇒ Hierachy Display で，どのファイルが作られたか見ることができる．同じプロジェクトのファイルは拡張子を除いて，みな同じ名前を持つ．ファイルの内容を見るには対応するイコンをクリックする．図 20で示されているプロジェクトは 5 つのファイルを含んでいる． myxor.vhd は VHDL ソース， myxor.rpt はコンパイル結果に関する情報を含んだレポートファイル， myxor.acf はデバス種類などの設定情報を含んだファイルである．

図 20: プロジェクトを構成するファイル [23]

4.6 シミュレーション

シミュレーションは設計した回路が正しく動作するかどうか確かめるための過程である．実際，回路に適当な入力 (テストベクトル) を与え，出力される値を予想値と比較する．シミュレーション方法は大きく分けて 2 つある． 1 つは，テスト信号を生成する回路 (テストベンチ) を VHDL で記述し，このテストベンチに設計した回路を接続する方法である．テストベンチは入出力ポートを持つことはできない (それの” 入力端子” と” 出力端子” は全部シミュレートしたい回路の” 出力端子” と” 入力端子” に継っているから）．この方法は PeakVHDL で採用されている．もう 1 つのシミュレーション方法は，どのような入力をどのように (つまりどの時刻にどの端子に) 与え，どの出力を見るかという情報を SCF ファイル (simulator channel file) に保存しておいて，シミュレートする時にシミュレータにこのファイルを読み込ませる方法

[22] ファイル名 = ./image/max/messg.ps

[23] ファイル名 = ./image/max/hir-dis.ps

である． **MAX+plus II** や Cadence 社の LeapFrog 等はこの方法を採用している．

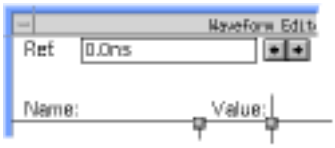


図 21: 波形エディタ

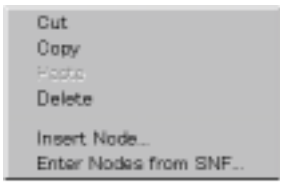


図 22: 信号を追加するためのショートカットメニュー

表 6: myxor の真理値表

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

MAX+plus II で SCF ファイルを作るには， **MAX+plus II** ⇒ Waveform Editor で波形エディタ (図 21) を起動する．

設計した myxor.vhd は表 6に示されているような真理値表を持つ排他的論理和回路である．この回路には x, y と z という 3 つの入出力端子があるから，まず SNF ファイルにこれらの信号を追加する．

波形エディタの中にマウスポインタを置き，右クリックすれば信号を追加するためのショートカットメニューが (図 22) 表示される．このショートカットメニューの Enter Nodes fromSCF... を選ぶと信号を選択するためのダイアログ (図 23) が現れる．キーボードで直接 *Node/Group* に信号名を入力することもできるが，スペルミスを侵しやすい．

List をクリックすれば左側に追加可能な信号が表示される．左クリックで信号を選択してから，⇒ をクリックすると選択した信号が右側にコピーされる．次に OK ボタンをクリックすれば右側にある信号が波形エディタウィンドウに追加される．波形エディタウィンドウにある信号を消したい場合は，その信号を右クリックし，ショートカットメニューの Cut を選択すればよい．

x, y と z を SNF ファイルに追加し終ったら，入力信号である x と y の波形を編集する (当然出力は入力により決まるから編集できない)．

図 23: 信号を選択するためのダイアログ [24]

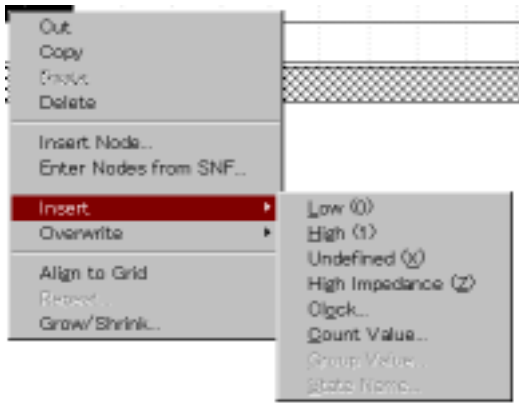


図 24: 波形を編集するためのショートカットメニュー [25]

信号の波形をドラッグするとドラッグした領域の色が反転する．この領域を右クリックし，ショートカットメニュー (図 24) を開き，Insert または Overwrite のサブメニューから，適当な項目を選ぶことにより，波形の反転した領域に値を設定できる．図 25のように x と y の値を設定する．

次に File ⇒ Save で作成した SCF ファイルを保存する．プロジェクトに SCF ファイルが自動的に追加されるのを確認することができる (方形波のアイコンが表示される)．

これで，シミュレーションに必要な情報は全部整っている． **MAX+plus II** ⇒ Simulator でシミュレータ (図 26) を起動し，Start ボタンをクリックすればシミュ

[24] ファイル名 = ./image/max/ennodes.ps

[25] ファイル名 = ./image/max/shortcut.ps

レーションが始まり， SCF の中に作成した出力信号の波形が計算される．計算した結果はすぐに波形エディタウィンドウに反映される．図 25から，出力 z は約 10ns の時間遅延を除けば，表 6の予想値と一致していることが分かる．

図 25: シミュレーション結果 [26]

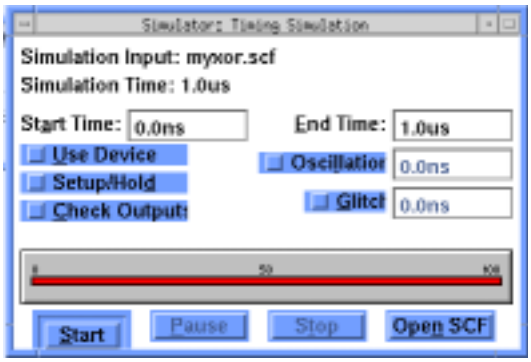


図 26: シミュレータウィンドウ [27]

4.7 書き込み用ファイルの作成

デバイス (FLEX チップ) の特定のピンに信号を送ることにより設計した回路をデバイスに書き込む訳であるが，この時必要になる情報 (特定のピンに送る信号列) を含んだファイルは **MAX+plus II** で生成できる．

デバイスの書き込み (コンフィグレーション) 方法により，このファイルの形式が異なるが，ここでは passive serial (外部の回路からデバイスに 1 ビットずつシリアルに書き込む）コンフィグレーションの場合に限って説明する．

コンパイラウィンドウを開いた状態で， File ⇒ Combine Programming Files を選択する．図 27のようなダイアログが画面に出てくる． Add をクリックして， myxor.sof を真中のボックスに追加する．ファイル形式として，*File Format* のプルダウンメ

[26] ファイル名 = ./image/max/simlre.ps

[27] ファイル名 = ./image/max/siml.ps

ニューから .ttf(Sequential) を選ぶ．続いて OK ボタンをクリックするとコンフィギュレーションに必要な情報を含んだ, myxor.ttf というファイルができる．

このファイルを実際のデバイスに書き込むには，付録 B に乗っている書き込みプログラムを使う．このプログラムは FLEX8000 のチップが付録 A の基板に乗っている場合を想定している． DOS プロンプトから

download myxor.ttf

を実行すれば myxor.ttf がデバイスに書き込まれる．

図 27: 書き込みファイルの作成 [28]

[28] ファイル名 = ./image/max/combine.ps

4.8 まとめ

以上で，MAX+plus II によるディジタル回路の設計から動作検証，デバイスの書き込みまで一通り説明した．以下に，全体の作業を表すフローチャートを図 28に示す．

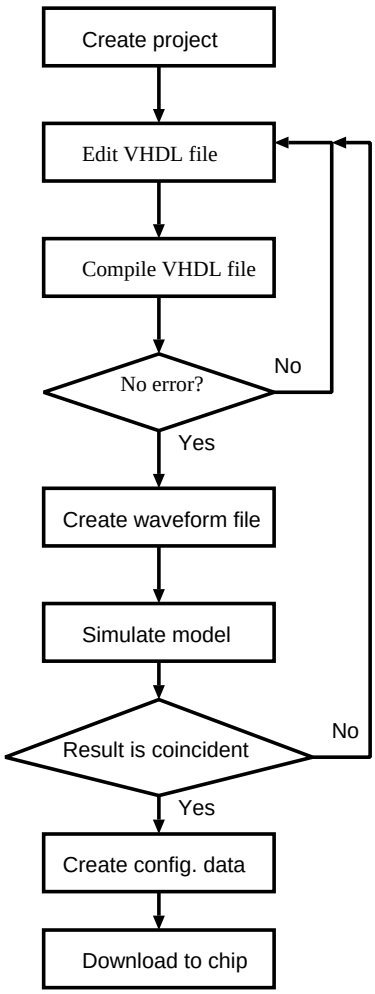


図 28: 設計フローチャート [29]

[29] ファイル名 = ./fig/max-de-f.eps

5 PeakVHDL&FPGA の使用方法

PeakVHDL&FPGA はアメリカの Accolade Design Automation 社 (以下 Accolade) が発売した VHDL 処理システムで，主に VHDL コンパイラと論理合成ツールから構成されている．VHDL コンパイラと全体のユーザインターフェイスは Accolade 社が開発したものであるが，論理合成ツールのエンジン部は Metamor 社のものを使っている．**PeakVHDL&FPGA** の主な機能を以下に列挙する．

- 設計モデルの階層構造を管理する部分
- VHDL ソースを編集するためのテキストエディタ
- VHDL ソースを構文解析しコンパイルするコンパイラ．コンパイラはシミュレータに必要な情報を生成する．バグがよくでる上，エラーについての説明も非常にいい加減である (コンパイラが表示できるエラーメッセージの約 8 割は”Syntax error”)．
- 設計した VHDL モデルの動作を検証するシミュレータ

から構成されている． PeakFPGA は PeakVHDL で記述した VHDL ファイルを実際のディジタル回路に変換するツールである．

MAX+plus II と同様に， **PeakVHDL&FPGA** も 1 つの設計に関連する全てのファイルを 1 つのプロジェクトとして管理する．従って，新しい設計は **MAX+plus II** と同じくプロジェクトの新規作成から始まる． **PeakVHDL&FPGA** では同じプロジェクトのファイルは同じディレクトリに作られるが．混乱しないように異なるプロジェクトを異なるディレクトリに保存するべきである．

5.1 VHDL ソースの作成

まず File ⇒ New Project を選択して，新しいプロジェクトを作る (ツールバーのボタン  をクリックしても同じことができる）．画面に図 29のようなプロジェクトウィンドウが表示される．このウィンドウには 4 つのボタンがある

1.  プロジェクト再構築．プロジェクトに新しいファイルを追加したり，設計の構成を変えたりする時に使う．

2.  プロジェクトの階層構造を表示する．後述するが，VHDL を用いた設計は階層構造を持つことができる．このボタンをクリックすればプロジェクト全体の依存関係が一目瞭然で分かる．

3.  プロジェクトの階層構造を隠す． 2 番目のボタンと逆の働きをする．

4.  ソース以外のファイルを全部消す．プロジェクトが多くのモジュールから構成されている場合，下位のモジュールを変更しても，その変更が上位のモジュールに反映されず，コンパイラがモジュールの依存関係を把握できなくなることがある．この時，ソース以外のファイルを全部消して，最初からコンパイルし直さなければならない．

図 29: プロジェクトウィンドウ [30]

次に File ⇒ New Module ... でプロジェクトに VHDL ソースを追加する．プロジェクトを保存するためのダイアログが表示されるが，プロジェクト名を入力して，保存をクリックする．今度は図 30にあるようなダイアログが現れる．このダイアログには 3 つのオプションがある．

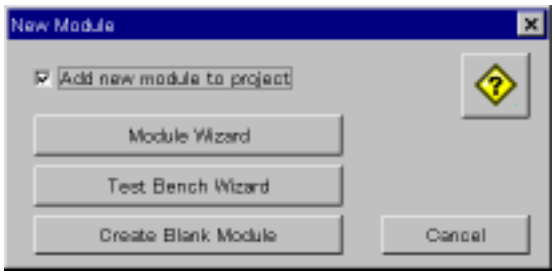


図 30: *New Module* ウィンドウ [31]

[30] ファイル名 = ./image/peak/prj-win.ps

[31] ファイル名 = ./image/peak/new-md-wz.ps

1. Module Wizard

   ポート名とポートのデータ型を入力するだけで **PeakVHDL&FPGA** が自動的に，**entity** や **architecture** 等の，どの VHDL ソースにもある決まった部分を含んだテプレートを生成してくれる．設計者は必要な部分を追加して VHDL ソースを完成する．この方法は，スペルミスはなく，入力量も軽減できるのでとても便利である．

2. Test Bench Wizard

   上のオプションと同じであるが，テストベンチを VHDL で記述する場合に使う．

3. Create Blank Wizard

   単にテキストエディタを起動する．設計者が VHDL ソースを全部入力する．

Create Blank Wizard をクリックして，前章で示した myxor.vhd の内容を入力しても同じ結果になるが，ここでは Module Wizard による方法を使う．Module Wizard ボタンをクリックすると **entity** 名，**architecture** 名とポートを入力するための *New Entity Wizard* ウィンドウ (図 31) が現れる．表 7のように入力する．

図 31: *New Entity Wizard* [32]

表 7: Entity,Architecture とポートの入力値

Entity Name	Architecture Name	Port Name	Mode	Type
myxor	Behavior	x	in	std_logic
		y	in	std_logic
		z	out	std_logic

入力が終ったら，Create をクリックし，ファイルを myxor.vhd という名前で保存してテキストエディタを起動する．**entity** , **architecture** , **library** 等の宣言が

[32] ファイル名 = ./image/peak/new-ent-wz.ps

自動的に挿入されているのが分かる．不要な部分を消して， **architecture** の **body** に

```
z <= x xor y ;
```

を追加して (前章と同じ VHDL ファイルを編集する)，ファイルを保存する．

上で行なったプロジェクトに対する変更をを PeakVHDL に知らせるためにプロジェクトウィンドウのボタン  をクリックする．MODULE MYXOR.VHD の前に + が表示される．この + をクリックすれば，プロジェクトの対応するモジュールの階層構造を見ることができる．

5.2 VHDL ソースのコンパイル

PeakVHDL&FPGA のコンパイラは **MAX+plus II** のコンパイラと違って，設計モデルを実際のデバイスに書き込むことを前提としないのでデバイスを指定する必要はない．また， VHDL ソースの論理合成は行なわず，ソースの構文解析とシミュレータに必要な情報を生成するだけである．

まず，プロジェクトウィンドウの MODULE MYXOR.VHD を左クリックでセレクトする．次に，ツールバーのボタン  をクリックする (または Simulate ⇒ Compile Selected を選択する) とコンパイルが始まる．同時に，メッセージを表示する *Transcript* ウィンドウ (図 32) も画面に出る．コンパイルの進行状況や発見されたエラー等の情報はこのウィンドウに出力される．

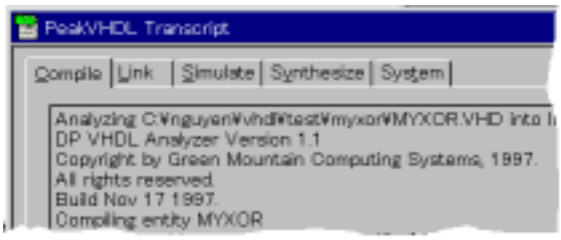


図 32: *Transcript* ウィンドウ [33]

[33] ファイル名 = ./image/peak/trans-win-cmp.ps

5.3 テストベンチの作成

MAX+plus II は SCF ファイルによりシミュレートしようとしているモデルに入力を与えるが，**PeakVHDL&FPGA** はテストベンチを使って，必要な入力を生成する．テストベンチとは入出力ポートを持たない特別な VHDL モデルである．後述するように，**wait** 文と信号代入文を組み合わせることにより VHDL で任意の (ディジタル) 波形を作り出せる．

シミュレートしたいモデルは **instantiation** 文によりテストベンチ内に下位モジュールとして組み込む[34]このように，シミュレーションを行なう時，プロジェクトの最上位モジュールは必ず，ポートを持たないテストベンチでなければならない．

テストベンチも前述の myxor.vhd と同様にして作れる．まず，プロジェクトウィンドウの MODULE MYXOR.VHD をクリックして，これをテストベンチに組み込むモジュールとして指定する．次に，ツールバーのボタン  またはメニュー File ⇒ New Module ... で *New Module* ウィンドウを開く．Test Bench Wizard をクリックして次に進む．ポートを入力するためのダイアログ (図 31) が表示される．表 7に示されている値を入力する．先に myxor をクリックしたから *Entity Name* にこの名前が自動的にコピーされる． *Entity Name* や *Port* 等はテストベンチのものではなく，シミュレートしたいモジュール (テストベンチに結合されるモジュール) のものであるという点に注意しなけらばならない．また，ポート名等を自分で入力すると間違えやすいので，myxor.vhd からポートの部分をコピーして *Port declarations* にペーストした方が良い．最後にボタン Create をクリックしてテキストウィンドウを起動する[35]．続いて，テストベンチを修正して，次のような内容に直す．

```
library ieee;
use ieee.std_logic_1164.all;

use std.textio.all;
use work.MYXOR;
```

[34]シミュレートしたいモデルを一個のチップに例えれば，テストベンチはチップを差し込むためのソケットを持つテスト基板で，**instantiation** はチップをソケットに差し込む操作に相当する．

[35]この時，**PeakVHDL&FPGA** がテストベンチのファイル名を聞いて来るが，関連が分かりやすいように，**PeakVHDL&FPGA** が生成する名前 (TEST_エンティティ名.VHD) を使うべきである．

```
entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component MYXOR is
    port (
        x: in std_logic;
        y: in std_logic;
        z: out std_logic
    );

end component;
constant PERIOD: time := 100 ns;
-- Top level signals go here...
signal x: std_logic;
signal y: std_logic;
signal z: std_logic;

begin
    DUT: MYXOR port map ( x,y,z );

    STIMULUS1: process
    begin
        x <= '0'; y <= '0'; wait for PERIOD;
        x <= '0'; y <= '1'; wait for PERIOD;
        x <= '1'; y <= '0'; wait for PERIOD;
        x <= '1'; y <= '1'; wait for PERIOD;
        wait;            -- Suspend simulation
    end process STIMULUS1;

end stimulus;
```

上の **STIMULUS1: process** と **end process STIMULUS1;** の間にあるのは myxor に与える入力 x と y を生成する部分である．このように記述すると，PERIOD(100ns の定数) 間隔で x と y に (0,0),(0,1),(1,0),(1,1) が順番に入ることになる (第 4章 – 図 25の波形ファイルと同じ入力).

5.4 シミュレーション

コンパイルできた時，設計した回路をシミュレートすることができる．前述したように，**PeakVHDL&FPGA** のコンパイラは実際のデバイスに書き込むことを前

提としないため，あくまで論理レベルでのシミュレーションしかできない (つまり，こういう入力に対してこういう出力がでるという論理値上の関係は分かるが，入力が変化してから出力が変化するまでの遅延時間等の物理的な関係は分からない）．遅延も考慮したシミュレーションをするには，他のソフトウェアで論理合成して時間情報を含んだ VHDL ソースを作っておく必要がある (この章の最後のセクション，**MAX+plus II** とのインターフェイスを参照).

設計した myxor をシミュレートするにはツールバーのボタン  でテストベンチをコンパイルしてから，ボタン  で myxor をこれにリンクする．コンパイルとリンクが終ったら，ボタン  またはメニュー Simulate ⇒ Load Selected でシミュレータを起動する．シミュレータはコンパイラにより作られた必要な情報をまず最初に読み込んむ．それから，変数・信号を選択するための *Select Display Objects* ウィンドウ (図 33) を開く．このウィンドウで選択した変数と信号だけがシミュレータ画面に表示される． myxor の場合は 3 つの信号 x, y, z しかないからこの操作はあまり意味を持たないが，一般の設計は変数・信号が非常に多く，すべての波形を表示すると乱雑になるので，必要なものだけを選ぶ必要がある．

図 33: *Select Display Objects* ウィンドウ [36]

Select Dislplay Objects ウィンドウの左側にある *Available objects* ボックスはシミュレータによって認識されている変数・信号を表示する．ここに何も表示されていなければ，コンパイルまたはリンクが正常に終了しなかったことを意味する. 右側の *Objects to display* ボックスにある変数・信号は波形がシミュレータ画面に表示されるものである. *Available objects* の変数・信号を左クリックで選んでから Add > をクリックすると選ばれたものが右側に移動する． Add Primaries をクリックすれば，入出力ポートだけが右側に移動する.

[36] ファイル名 = ./image/peak/sel-dis-obj.ps

Add > や Add Primaries 等で表示する変数・信号を指定してから，Close で *Select Display Objects* ウィンドウを閉じ，シミュレータ画面に進む．

図 34: シミュレータ画面のツールバー [37]

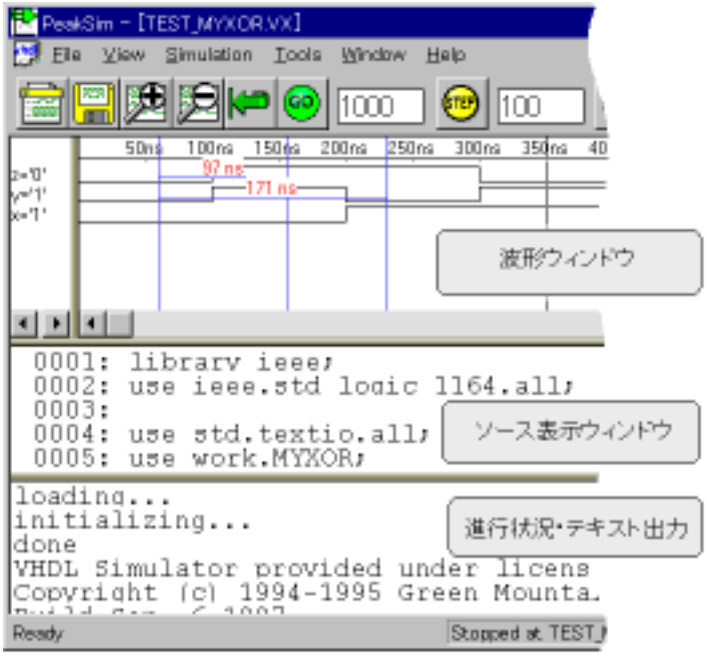


図 35: シミュレータウィンドウ [38]

シミュレータ画面は基本的に 4 つの部分から構成されている．一番上にあるのはツールバー (図 34) である．ツールバーの下にあるウィンドウはシミュレーション結果である変数・信号の波形と値を表示する (図 35)．真中にあるウィンドウはシミュレートしている VHDL ソースを表示する．ソースが 2 つ以上ある場合は上のコンボボックス TEST_MYXOR.VH で切替えることができる．一番下にあるウィンドウはシミュレーションの進行状況と VHDL の **writeline** 文によるテキスト出力を表示する．これらのウィンドウの大きさはいずれも境界線をドラッグすることにより変更できる．シミュレータ画面のツールバーのボタン GO でシミュレーションを開始する．

[37] ファイル名 = ./image/peak/sim-tb.ps

[38] ファイル名 = ./image/peak/sim-win.ps

シミュレーションが終ると一番上のウィンドウに波形が表示される．波形ウィンドウにマウスポインタを置けばカーソルと呼ばれる細い縦線が表示される．このカーソルはマウスの動きに追従して移動する．カーソルの位置における変数・信号の値は波形ウィンドウの左側に表示される．波形ウィンドウをクリックすると，クリックした位置に固定カーソルが表示される．固定カーソルを 2 本以上置くことにより時間間隔を測ることができる (図 35)．

図 35から分かるように，**PeakVHDL&FPGA** は時間遅延を考慮しないので，前章の **MAX+plus II** によるシミュレーション結果 (図 25) と違って，入力が変化すれば，その変化が遅延なくすぐに出力に反映される．

5.5 論理合成

論理合成 (synthesize) はハードウェア記述言語で記述したソースを実際のディジタル回路に変換する過程である． VHDL は **wait** 文や **float** 型等のように，本質的にシミュレーション向けで論理合成不可能または合成できても効率が非常に悪く，実用性がない部分を多く持っている．一方，論理合成はまだ発展途上の技術であるため，このような構文がなく， VHDL コンパイラを通って，かつ論理レベルでのシミュレーションができても，論理合成できない VHDL ソースが多くある．

PeakVHDL&FPGA の論理合成ツールは VHDL ソースの変換結果として EDIF(Electronic Design Interchange Format) ファイルを出力する． EDIF はディジタル回路を表す業界標準のファイル形式で， EDIF ファイルの中に，使用する基本部品 (セル) の定義とこれらのセルの相互接続関係がテキストフォーマットで書かれている．どのようなセルが使えるかは， EDIF ファイルを処理するツールに依存するので，あらかじめそれを指定しなければならない．使用するセルのセットを指定するには Synthesize ⇒ Options を選択する．図 36のようなオプション設定ウィンドウが表示される．以下はよく使うオプションである．

1. *Library Mapping* VHDL ソースの中に算術演算などのライブラリによりサポートされる記述があれば，対応するライブラリをチェックする．

2. *Device Family* このオプションでセルセットを指定する．**MAX+plus II** で EDIF ファイルを処理する場合は *Altera,all devices(EDIF)* を選ぶ．

3. *Analyze only* 論理合成ツールは VHDL ソースを実際のディジタル回路に変換する前に，そのソースを構文解析し，ソースが変換可能かどうかを確かめる．普通このプロセスに比べて，実際の変換過程の方が多くの時間がかかる．ソースの構文をチェックする，あるいは単にソースが論理合成可能かどうかを知りたい場合はこのオプションをチェックする．

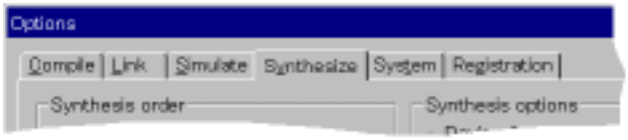


図 36: オプション設定ウィンドウ [39]

ここでは (myxor の場合)， *Device Family* だけを *Altera, all devices(EDIF)* に指定し，他のオプションはデフォルト値を使う．

OK ボタンでオプションウィンドウを閉じ，ツールバーの  で論理合成を開始する．進行過程やエラーメッセージ等は VHDL ファイルと同じディレクトリに Metamor.log という名前で保存される．このファイルは Synthesize ⇒ View Synthesis Log で見ることができる．

5.6 MAX+plus II とのインターフェイス

MAX+plus II の Interfaces メニューのオプション EDIF Netlist Reader を選択してオンにすれば，**PeakVHDL&FPGA** で作成した EDF ファイルを読み込んで処理することができる．また，同じ Interfaces メニューの VHDL Netlist Writer をオンにすれば，**MAX+plus II** の図形入力ツールや AHDL などで設計した回路も VHDL ソース変換することができる．

PeakVHDL&FPGA では，時間遅延を考慮したシミュレーションは直接行なうことができないと述べたが，この VHDL Netlist Writer 機能を使えば，間接的にできる．つまり，**PeakVHDL&FPGA** の VHDL ファイルを論理合成して EDF ファ

[39] ファイル名 = ./image/peak/options.ps

イルを作り，MAX+plus II で EDF ファイルを，PeakVHDL&FPGA がシミュレートできる，遅延情報を含んだ VHDL ファイルに変換するのである．

6　VHDL による行列個有ベクトル計算モデル

本章はまず簡単な例を通して，ソフトウェア的なプログラムがどのようにハードウェア化 (実際のディジタル回路で実現) されるかという一般的な設計方法について述べる．次に，作成した行列固有ベクトルの計算モデルについて具体的に説明する．

6.1　ソフトウェア的なアルゴリズムの VHDL 記述

ソフトウェア的なプログラムはステートマシンによるモデルでハードウェア化できる．まず，実行中そのプログラムがどのような状態を持ち，それぞれの状態にある時，プログラムがどのような処理を行なうか，割り出さなければならない．例えば，プログラムが $s_1, s_2, ..., s_n$ の n 状態を持ち，各状態に置ける処理はそれぞれ $p_1, p_2, ..., p_n$ であるとすると，このプログラムと同等な動きをするハードウェアは VHDL で以下のように記述できる．

```
P1 :  process begin
    if rising_egde(Clock) then
      State <= NextState;
    end if ;
end process P1;


P2 :  process begin
      if falling_edge(Clock) then
      case State is
          when s_1 => NextState <= s_2; p_1 を実行;
          when s_2 => NextState <= s_3; p_ を実行;
         ....


      end case ;
    end if ;
end process P2;
```

つまり，クロックの立ち上がりに同期してハードウェアの状態が変わり，クロックの立ち下がりで，現在の状態に対応する処理を行うと同時に，次の状態を決定するのである．[40]

例として， 整数 n を入力すると $\sum_{i=1}^{n} i$ を出力する計算機を考える．これと同等なソフトウェアプログラムは，例えば BASIC で書くと次のようになる．

```
 0 sum = 0
10 for I=1 to n
20 sum = sum +I
30 next i
```

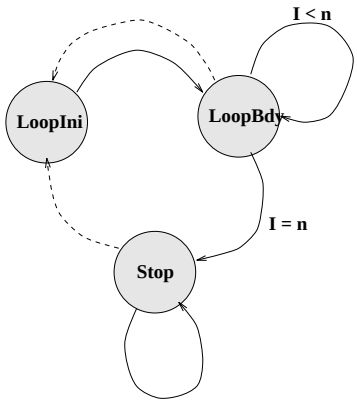


図 37: ステートマシンの状態遷移図 [41]

設計している計算機は 3 つの状態を持っていると考えることができる．

- LoopIni： 初期化状態．この状態では，出力 Sum と ループ変数 I に初期値を代入する．この状態の次は LoopBdy であるから， NextState = LoopBdy
- LoopBdy ： 計算状態． Sum に I を足し， I を 1 つ増やす．これの次の状態は， I が n より小さければ同じ LoopBdy であり，そうでなければ Stop である．
- Stop : 最終状態である．リセットされなければ，次の状態もまたこの状態である．

[40] 上記は見通しがよくなるように 2 つのプロセスに分けているが，普通，順序回路を含む部分 (**if** rising_egde(Clock) など) と組み合わせ回路を含む部分 (式の代入など) を別々のプロセスに分けた方が，論理合成効率がよくなる．

[41] ファイル名 = ./fig/state-machine.xfig.eps

この計算機の状態遷移は図 37のようになる．クロックによる同期遷移は矢印のついている実線で表されている．矢印線の横にあるのは遷移条件で，この条件が成り立っている時に，クロックが入ったら遷移が起こるということを意味する．点線はリセット信号による非同期遷移を表している．計算機はどんな状態にあっても，リセット信号が入るとただちに最初の状態 LoopInini に戻る． n を入力し，リセットしてから $\sum_{i=1}^{n}$ が求まるまで n クロックが必要である．

この計算機の完全な VHDL モデルを以下に示す．

```
library ieee;
use ieee.std_logic_1164.all;

entity EXAMPLE is
 port (
   Clock,Reset : in std_logic;
   n           : in integer;
   Sum         : out integer;
   );
end EXAMPLE

architecture BEHAVIOR of EXAMPLE is
  type STATE_TYPE is ( LoopIni, LoopBdy, Stop);
  -- 状態を表す新しいデータ型を宣言
  signal State,NextState : STATE_TYPE;
  signal I               : integer;
begin
  P1 : process(CLK,Reset,NextState)
  begin
       if( Reset='1') then -- リセットが優先する
           State <= LoopIni; -- リセットされると直ちに初期状態に戻る
       elsif rising_egde(Clock) then
           State <= NextState; -- 状態遷移はクロックに同期する
       end if;
  end process P1;

  P2 : process(CLK,State,NextState,n,I)
  begin
       if falling_egde(Clock) then
           case State is
               when LoopIni =>
                  I <= 1;  Sum <= 0;
                  NextState <= LoopBdy;
               when LoopBdy =>
                  Sum <= Sum + I;
                  if( I < n ) then
                     -- 次も同じ状態であるから
```

```
                    --NextState を変更する必要はない
                    I <= I+1;
                  else
                    NexState <= Stop;
                  end if;
               when others  =>
                  NextState <= Stop; -- Null でも良い
            end case;
      end if;
  end process P2;
end BEHAVIOR;
```

6.2　行列固有ベクトルの計算モデル

前述したように，VHDL によるハードウェア化は結局，全体の複雑な計算過程を簡単なブロック (四則演算レベルの基本的な部分) に細かく振り分け，各々のブロックに 1 つの状態を割り当てた後，状態間の遷移と各状態における計算を記述する作業に帰着する．

ここでは，第 2章で述べたアルゴリズムがどのような基本ブロックに分けられ，そしてこれらのブロックが，VHDL によるステートマシンモデルのどの状態に対応しているのかを示す．完全な VHDL ソースは付録に掲載してある．

6.2.1　構成について

固有ベクトルを求めたい行列 A と，求めた固有ベクトルを保存するために大きさ n^2 の配列を 2 つ使う．他に，固有値と中間的な数値を保存するためのメモリも幾つか用意する．ハウスホルダ変換にでてくるベクトル $\mathbf{w}$ は行列 A の下半分に当たる領域に書き込まれる．必要なメモリの大きさはほぼ $2 \times n^2 \times$ (浮動小数点数のバイト数)．例えば，次数 5000 の行列で，倍精度を使う場合は $2 \times 5000^2 \times 8 \cong 400$ Mbyte となる．

付録 A.2の VHDL ソース中に使われている信号は表 8と表 9にまとめてある．表 8にあるのは計算用信号で，第章 2で説明したアルゴリズムと直接関係のあるものである．表 9にあるのは，クロックやループ用カウンタ等の，専用計算機を制御するため信号である．また，ここで便宜のため以下のような新しいデータ型を定義した．

- SCALAR 浮動小数点数を表している．現時点では，シミュレーションできる

ように，VHDL の **real** 型を用いているが，論理合成をする場合は，適切な型 (**std_logic_vector** 等) に再定義する必要がある.

- MEM_TYPE 大きさ n^2 の，SCALAR の配列
- VECTOR 大きさ n の，SCALAR の配列
- COUNTER ループ用カウンタのデータ型.
- STATE_TYPE 計算機の状態を表す列挙型．計算機の動作の密接な関係があるので 6.2.2で詳しく述べる.

表 8: 計算用信号

信号名	対応する数値	型	必要なメモリ領域 (バイト)
Mem	n × n 行列 A	MEM_TYPE	$n^2 \times$ b [42]
X	固有ベクトル (n 個)	MEM_TYPE	$n^2 \times$ b
Lamda	固有値 (n 個)	VECTOR	n × b
s2	$s^2 = \sum_{i=k+1}^{n}$	SCALAR	b
s	$s = \sqrt{s}$	SCALAR	b
HsC	$c_k = \frac{1}{s^2+s\times a_{k+1}}$(n 個)	VECTOR	n × b
P	ベクトル $\mathbf{p} = c_k A\mathbf{w}$	VECTOR	n × b
tmp	$tmp = \frac{c}{2}\mathbf{p}^T\mathbf{q}$	SCALAR	b
Q	ベクトル $\mathbf{q} = \mathbf{p} - tmp\mathbf{w}$	VECTOR	n × b
Alfa	三重対角行列の主対角成分	VECTOR	n × b
Beta	三重対角行列の副対角成分	VECTOR	n × b
Max	Gerschgorin 定理による初期範囲	SCALAR	b
A	二分法における固有値の下限	SCALAR	b
B	二分法における固有値の上限	SCALAR	b
C	$c = \frac{a+b}{2}$	SCALAR	b
a_mI	$A - \mu\mathbf{I}$ の主対角成分	VECTOR	n × b

表 9: 制御用信号

信号名	意味	型
clock	クロック	std_logic
reset	リセット	std_logic
State	計算機の状態	STATE_TYPE
i,j,k	ループ変数	COUNTER

6.2.2　動作について

ここで設計した専用計算機モデルの動作は，第 2で述べたアルゴリズムに対応して 4 つフェーズに分けることができる.

ハウスホルダ変換　行列 A を三重対角行列に変換する．変換が終了した後，A の下半分に **w** が保存される.

状態名	この状態に対する動作
HsPrnDat	行列 A の各成分を表示
HsKLoopIni	第 k 段階の変換の初期設定
Hs2LoopIni,Hs2LoopBdy	$s^2 = \sum_{i=k+1}^{n}$ を求める
HsSCal,HsCCal,HsViceCal	$s = \sqrt{s^2}$ と $c = \frac{1}{s^2 + s \times a[k][k+1]}$ を求める
HsPLoopIni,HsPLoopBdy	$\mathbf{p}$ を求める
HsPtxWLoopIni, HsPtxWLoopBdy,	$\mathbf{p}^T\mathbf{w}$ を求める
HsTmpCal,HsQLoopIni,HsQLoopBdy	$\mathbf{q} = \mathbf{p} - tmp\mathbf{w}$ を求める
HsALoopIni, HsALoopBdy	$A_{k+1} = A_k - (\mathbf{p}\mathbf{q}^T - \mathbf{q}\mathbf{p}^T)$
HsPrnRes	変換結果を表示

二分法　固有値を求め，Lamda に入れる.

状態名	この状態に対する動作
BisSetDat, BisPrnDat	ハウスホルダ変換をした行列 A から対各成分を抽出する
BisMaxLoopIni, BisMaxLoopBdy	Gerschgorin 定理により最大範囲を求める
BisEvLoopIni,BisEvEnd	n 個の固有値を求めるループ
BisBisLoopIni	固有値の存在範囲を 2 分していく
BisCCal,BisSturmLoopIni, BisSturmLoopBdy	c より大きい固有値の個数を求める.
BisNewRange	新しい範囲を設定
BisPrnRes	求めた固有値を出力

逆反復法　三重対角行列の固有ベクトルを求め，X に保存する.

状態名	この状態に対する動作
IitKLoopIni,IitKLoopEnd	n 個の固有ベクトルを計算するループ
IitMuCal,IitA_muILoopIni, IitA_muILoopBdy	$A - \mu\mathbf{I}$ を計算する
IitXIniLoopIni, IitXIniLoopBdy	固有ベクトルを初期化する
IitSlvMACalLoopIni, IitSlvMACalLoopBdy	$A - \mu\mathbf{I}$ をLU 分解する
IitSlvForwLoopIni, IitSlvForwLoopBdy	前進消去
IitSlvBackLoopIni, IitSlvBackLoopBdy	後進消去
IitPrnRes	結果を出力

逆変換　三重対角行列の固有ベクトルと A の下半分に入っている **w** ベクトルを用いて，元の行列 A の固有ベクトルを求める.

状態名	この状態に対する動作
ITrjLoopIni, ITrjLoopEnd	n 個の固有ベクトルを計算するループ
ITrkLoopIni, ITrkLoopEnd	$(\mathbf{I} - c\mathbf{w}_k\mathbf{w}_k^T)\times\mathbf{x}$ を求める
ITriLoop1Ini, ITriLoop1Bdy	$tmp = \mathbf{w}_k^T \times \mathbf{x}$ を求める
ITriLoop2Ini, ITriLoop2Bdy	$\mathbf{x} - tmp\mathbf{x}$ を求める

規格化　固有ベクトルの絶対値が 1 なるように固有ベクトルをスカラ倍する．この操作はなくても良い．

状態名	この状態に対する動作
NormILoopIni,NormILoopEnd	n 個の固有ベクトルを処理するループ
NormJLoop1Ini, NormJLoop1Bdy	固有ベクトルのを求める
NormJLoop2Ini, NormJLoop2Bdy	各成分を絶対値で割る
NormPrnRes	
STOP	終了

7　シミューレション結果と残された課題

以下は作成した計算モデルのシミュレーション結果を示す.

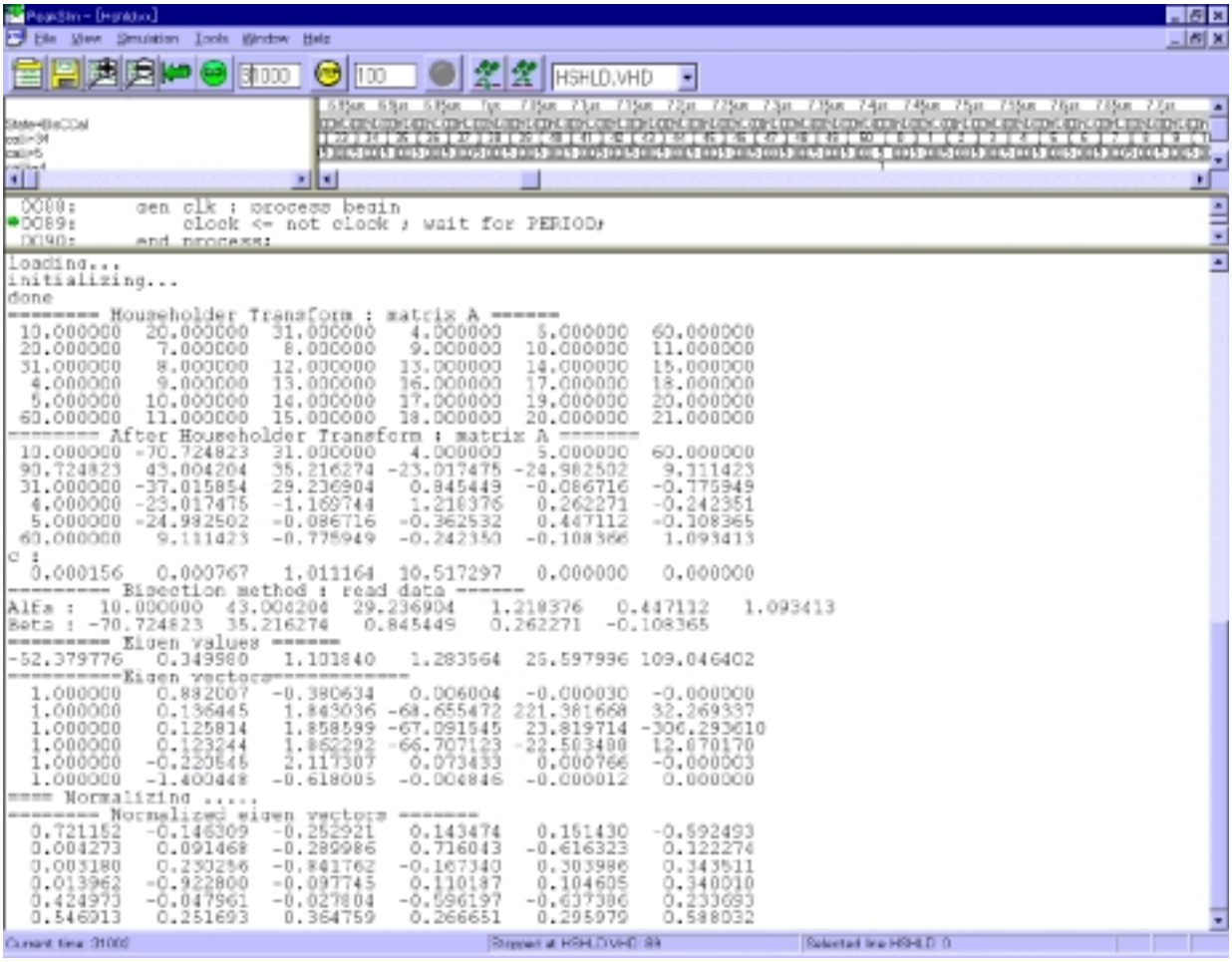


図 38: 固有ベクトル計算モデルのシミュレーション結果 [43]

比較するためにここに, C プログラムによる計算結果も示す.

$$6\times6\text{行列}A = \begin{pmatrix} 10 & 20 & 31 & 4 & 5 & 60 \\ 20 & 7 & 8 & 9 & 10 & 11 \\ 31 & 8 & 12 & 13 & 14 & 15 \\ 4 & 9 & 13 & 16 & 17 & 18 \\ 5 & 10 & 14 & 17 & 19 & 20 \\ 60 & 11 & 15 & 18 & 20 & 21 \end{pmatrix}$$

[43] ファイル名 = ./image/sim-re/sim-hshld.ps

$$\text{固有値}\lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix} = \begin{pmatrix} -52.3798 \\ 0.34998 \\ 1.10184 \\ 1.28356 \\ 25.598 \\ 109.046 \end{pmatrix} \quad ,\text{固有ベクトル} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \\ \mathbf{x}_4^T \\ \mathbf{x}_5^T \\ \mathbf{x}_6^T \end{pmatrix}$$

$$= \begin{pmatrix} -0.72115 & 0.14631 & 0.25292 & -0.14347 & -0.15143 & 0.59249 \\ 0.0042731 & 0.091469 & -0.28999 & 0.71604 & -0.61632 & 0.12227 \\ -0.00318 & -0.23025 & 0.84176 & 0.16734 & -0.30399 & -0.34351 \\ -0.013962 & 0.9228 & 0.097745 & -0.11019 & -0.10461 & -0.34001 \\ 0.42497 & -0.047961 & -0.027804 & -0.5962 & -0.63739 & 0.23369 \\ -0.54691 & -0.25169 & -0.36476 & -0.26665 & -0.29598 & -0.58803 \end{pmatrix}$$

上の数値結果から，作成したモデルが正しく動作することが分かる．

謝辞

本研究及び論文作成に当たり、終始御懇切なる御指導、御鞭撻を賜わりました指導教官の齋藤理一郎助教授に衷心より御礼の言葉を申しあげます。

また、本研究を進めるにあたり、熱心な御指導をいただくとともに種々の御高配を賜わりました木村忠正教授、湯郷成美助教授に深謝の意を表します。

また、研究活動をともにし、多くの援助をいただいた松尾竜馬さんに深謝いたします。

そして、数々の御援助、御助言をしていただいた竹谷隆夫さん、八木将志さん，はじめ木村・齋藤研究室の大学院生、卒研生の方々，高田氏をはじめ (株) 画像技研の方々に感謝します。

A　VHDL による行列固有ベクトル計算モデル

A.1　2 乗根の VHDL ソース

第 2章で説明したように行列の固有ベクトルを計算するために，実数の四則演算と 2 乗根が必要である．実数の四則演算は VHDL の中で予め定義されているが，2 乗根は定義されていない．以下は本研究が使用した 2 上根の VHDL ソース (Newton–Raphason の近似法を用いている) である．

```
-- filename  : ./source/vhdl/math.vhd
-- created   : Thu Feb 15 1998
-- author    : N.M.Duc
-- purpose   : root square package

Library IEEE;

Package MATH_REAL is
    function SQRT (X : real ) return real;
     -- returns square root of X;  X >= 0
end  MATH_REAL;



Package body MATH_REAL is
    function SQRT (X : real ) return real is
     -- returns square root of X;  X >= 0
--
-- Computes square root using the Newton-Raphson approximation:
-- F(n+1) = 0.5*[F(n) + x/F(n)];
--
constant inival: real := 1.5;
constant eps : real := 0.000001;
constant relative_err : real := eps*X;

variable oldval : real ;
variable newval : real ;

    begin
-- check validity of argument
if ( X < 0.0 ) then
assert false report "X < 0 in SQRT(X)"
severity ERROR;
return (0.0);
end if;

-- get the square root for special cases
if X = 0.0 then
   return 0.0;
else
```

```
if ( X = 1.0 ) then
return 1.0; -- return exact value
end if;
end if;

-- get the square root for general cases
oldval := inival;
newval := (X/oldval + oldval)/2.0;

while ( abs(newval -oldval) > relative_err ) loop
oldval := newval;
newval := (X/oldval + oldval)/2.0;
end loop;

return newval;
    end SQRT;
End MATH_REAL;
```

A.2　行列固有ベクトルの計算モデル (本体)

```
--file name : ./source/vhdl/hshld.vhd
--created   : Thu Jan 29th 1998
--author    : N.M.Duc
--purpose   : calculate eigen vectors using Householder transform

library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use ieee.std_logic_textio.all;

use work.math_real.all;


entity HSHLD is
end HSHLD;

architecture BEHAVIOR of HSHLD is
   type STATE_TYPE is (--- Householder Transform
                HsPrnDat,
                HsKLoopIni,
                HsS2LoopIni, HsS2LoopBdy, HsSCal, HsCCal,
                HsViceCal,
                HsPLoopIni, HsPLoopBdy,
                HsPtxWLoopIni, HsPtxWLoopBdy, HsTmpCal,
                HsQLoopIni, HsQLoopBdy,
                HsALoopIni, HsALoopBdy,
                HsPrnRes,
                --- Bisection method
```

```
                BisSetDat, BisPrnDat,
                BisMaxLoopIni, BisMaxLoopBdy,
                BisEvLoopIni, BisBisLoopIni,
                BisCCal,BisSturmLoopIni, BisSturmLoopBdy,
                BisNewRange,
                BisEvEnd,BisPrnRes,
                --- Inverse iteration
                IitKLoopIni,
                IitMuCal,IitA_muILoopIni, IitA_muILoopBdy,
                IitXIniLoopIni, IitXIniLoopBdy,
                IitSlvLoopIni, IitSlvCpyAlfLoopIni,
                IitSlvCpyAlfLoopBdy,
                IitSlvMACalLoopIni, IitSlvMACalLoopBdy,
                IitSlvForwLoopIni, IitSlvForwLoopBdy,
                IitSlvBackLoopIni, IitSlvBackLoopBdy,
                IitSlvLoopEnd,
                IitKLoopEnd, IitPrnRes,
                --- Inverse transform
                ITrjLoopIni, ITrkLoopIni,
                ITriLoop1Ini, ITriLoop1Bdy,
                ITriLoop2Ini, ITriLoop2Bdy,
                ITrkLoopEnd, ITrjLoopEnd,
                --- Normalize
                NormILoopIni,
                NormJLoop1Ini, NormJLoop1Bdy,
                NormJLoop2Ini, NormJLoop2Bdy,
                NormILoopEnd, NormPrnRes,

                STOP );
type MEM_TYPE is array (0 to 399) of real;
     -- largest dimension n is 20
type VECTOR   is array (0 to 19) of real;
subtype SCALAR is  real;
subtype COUNTER is integer range 0 to 100;

constant PERIOD    : time := 2 ns;
constant BISEC_R   : integer := 50;
constant INVIT_D   : real := 500.0;
constant INVIT_R   : integer := 10;

signal Mem         : MEM_TYPE; -- matrix A
signal X           : MEM_TYPE; -- eigen vectors
signal clock,reset : std_logic := '0';
signal State       : STATE_TYPE;

signal s2,s        : SCALAR;
signal Q,HsC       : VECTOR;

signal Alfa,Beta   : VECTOR;
```

```
   signal Max,A,B,BisC: SCALAR;
   signal Lamda       : VECTOR;

   signal IitXini     : VECTOR;
   signal a_mI        : VECTOR;
   signal a_mI_var    : VECTOR;

   signal ITrR        : SCALAR;
begin
    -- generate clock pulse
   gen_clk : process begin
      clock <= not clock ; wait for PERIOD;
   end process;

   -- generate reset pulse
   gen_reset : process begin
      reset <= '1'; wait for PERIOD/2;
      reset <= '0'; wait;
   end process gen_reset;

   -- execute Householder transform
   cal:process(clock,reset)
      variable i,j,k       : COUNTER;
      variable P           : VECTOR;
      variable tmp,G       : real;
      variable nBigger     : integer;
      variable IitM        : VECTOR;

      file data_file       : std.textio.text is  "hshld.dat";
      file result_file     : std.textio.text is  "result.dat";
      variable Lin,Lout    : line;
      variable data        : real;
      variable space       : character;
      variable N           : integer;

   begin
       -- restart and read data from file when reset is rising up
      if(reset='1') then
         readline(data_file,Lin);
         read(Lin,N);

           for ii in 0 to N-1 loop
              readline(data_file, Lin );
              for jj in ii to N-1 loop
                  read (Lin, data); read(Lin,space);
                  Mem(ii*N+jj) <= data;
                  Mem(jj*N+ii) <= data;
                  --write(Lout,data); write(Lout," ");
              end loop;
```

```
        --writeline(output,Lout);
     end loop;
     IitXIni(0) <= 119.0; IitXIni(1) <= 1.0; IitXIni(2) <= 2.0;
     IitXIni(3) <= 3.0; IitXIni(4) <= 4.0; IitXIni(5) <= 445.0;
     IitXIni(6) <= 6.0; IitXIni(7) <= 7.0; IitXIni(8) <= 8.0;
     IitXIni(9) <= 9.0; IitXIni(10) <= 10.0; IitXIni(11) <= 11.0;

    State <= HsPrnDat;
else
    if(rising_edge(clock)) then
       case State is
          when HsPrnDat =>
             write(Lout,"==== Householder Transform : matrix A ======");
             writeline(output,Lout);
                  for ii in 0 to N-1 loop
                    for jj in 0 to N-1 loop
                      write(Lout,Mem(ii*N+jj),right,10); write(Lout,' ');
                    end loop;
                    writeline(output,Lout);
                  end loop;
             State <= HsKLoopIni;

          when HsKLoopIni =>
             k := 0;
             State <= HsS2LoopIni;

          when HsS2LoopIni =>
             s2 <= 0.0;
             j := k+1;
             State <= HsS2LoopBdy;

          when HsS2LoopBdy =>
             s2 <= s2+ Mem(j*N + k)*Mem(j*N+k);
             if(j < N-1) then
                j := j+1;
             else
                State <= HsSCal;
             end if;

          when HsSCal =>
             if( Mem((k+1)*N+k)>0.0) then
                s <= sqrt(s2);
             else
                s <= - sqrt(s2);
             end if;
             State <= HsCCal;

          when HsCCal =>
             HsC(k)  <= 1.0/( s2 + Mem((k+1)*N+k)*s);
```

```
   State <= HsViceCal;

when HsViceCal =>
   Mem(k*N+k+1) <= -s;
   Mem((k+1)*N+k) <= Mem((k+1)*N+k) + s;
   State <= HsPLoopIni;

when HsPLoopIni =>
   i := k;
   j := k+1;
   P(i) := 0.0;
   State <= HsPLoopBdy;
when HsPLoopBdy =>
  --- calculate each element of vector P ( j=k to N-1)
  --test  := Mem1((i-1)*N+j);
  --test1 := W(j);
  P(i) := P(i) + Mem(i*N+j) * Mem(j*N+k);
  if(j<N-1) then
     j := j+1;
  else
     P(i) := HsC(k)*P(i);
     if(i<N-1) then -- go to next loop
        i := i+1;
        j := k+1;
        P(i) := 0.0;
     else
         State <= HsPtxWLoopIni;
      end if;
   end if;

when HsPtxWLoopIni =>
   tmp := 0.0;
   i := k+1;
   State <= HsPtxWLoopBdy;

when HsPtxWLoopBdy =>
   tmp := tmp + P(i)*Mem(i*N+k);
   if( i<N-1) then
      i := i+1;
   else
      State <= HsTmpCal;
   end if;

when HsTmpCal =>
   tmp := tmp*HsC(k)/2.0;
   State <= HsQLoopIni;

when HsQLoopIni =>
   i := k+1;
```

```
                Q(k) <= P(k);
                State <= HsQLoopBdy;
             when HsQLoopBdy =>
                Q(i) <= P(i) - tmp*Mem(i*N+k);
                if(i<N-1) then
                   i := i+1;
                else
                   State <= HsALoopIni;
                end if;

             when HsALoopIni =>
                i := k+1;
                j := k+1;
                State <= HsALoopBdy;
             when HsALoopBdy =>
                Mem(i*N+j) <= Mem(i*N+j)-Mem(i*N+k)*Q(j)-Mem(j*N+k)*Q(i);
                if(j<N-1) then
                   j := j+1;
                else
                   if(i<N-1) then
                      j := k+1;
                      i := i+1;
                   else
                      if(k<N-3) then
                         k := k+1;
                         State <= HsS2LoopIni;
                      else
                         State <= HsPrnRes;
                      end if; -- of k loop
                   end if;
                end if;

             when HsPrnRes =>
                write(Lout,
                  "====== After Householder Transform : matrix A =======");
                writeline(output,Lout);
                for ii in 0 to N-1 loop
                   for jj in 0 to N-1 loop
                     write(Lout,Mem(ii*N+jj),right,10); write(Lout,' ');
                   end loop;
                   writeline(output,Lout);
                 end loop;
                 write(Lout,"c :"); writeline(Output,Lout);
                 for ii in 0 to N-1 loop
                    write(Lout,HsC(ii),right,10); write(Lout," ");
                 end loop;
                 writeline(output,Lout);
                 State <= BisSetDat;
----------  end of Householder Transform -----------------------------
```

```
 when BisSetDat =>
     for ii in 0 to N-1 loop
      Alfa(ii) <= Mem(ii*N+ii);
   end loop;
   for ii in 0 to N-2 loop
       Beta(ii) <= Mem(ii*N+ii+1);
   end loop;
   State <= BisPrnDat;

  when BisPrnDat =>
   write(Lout,
        "========= Bisection method : read data ======");
   writeline(output,Lout);
   write(Lout,"Alfa : ");
   for ii in 0 to N-1 loop
      write(Lout,Alfa(ii),right,10);
      write(Lout," ");
   end loop;
   writeline(output,Lout);
   write(Lout,"Beta : ");
    for ii in 0 to N-2 loop
      write(Lout,Beta(ii),right,10);
      write(Lout," ");
   end loop;
   writeline(output,Lout);
   State <= BisMaxLoopIni;

when BisMaxLoopIni =>
   Max <= abs(Alfa(0))+abs(Beta(0));
   i := 1;
   State <= BisMaxLoopBdy;

when BisMaxLoopBdy =>
   if(i<N-1) then
      tmp := abs(Alfa(i))+abs(Beta(i))+abs(Beta(i-1));
   else
      tmp := abs(Alfa(i))+abs(Beta(i-1));
   end if;
   if(Max <tmp) then
      Max <= tmp;
   end if;
   if(i<N-1) then
      i := i+1;
   else
      State <= BisEvLoopIni;
   end if;

when BisEvLoopIni =>
   k := 1;
```

```
   State <= BisBisLoopIni;

when BisBisLoopIni =>
   A <= - Max;  B <= Max;
   i := 0;
   State <= BisCCal;

when BisCCal =>
    BisC <= (A+B)/2.0;
    State <= BisSturmLoopIni;

when BisSturmLoopIni =>
   G := BisC - Alfa(0);
   if (G<0.0 or G=0.0) then
      nBigger := 1 ;
   else
      nBigger := 0;
   end if;
   j := 1;
   State <= BisSturmLoopBdy;

when BisSturmLoopBdy =>
   if(G=0.0) then
      j := j+1;
      G := BisC - Alfa(j);
      if ( G <0.0 or G =0.0) then
         nBigger := nBigger+1;
      end if;
   else
      G := BisC - Alfa(j) - Beta(j-1)*Beta(j-1)/G;
      if ( G<0.0 or G=0.0) then
         nBigger := nBigger+1;
      end if;
   end if;

   if(j<N-1) then
      j := j+1;
   else
      State <= BisNewRange;
   end if;

when BisNewRange =>
   if nBigger > (N-k) then
      A <= BisC;
   else
      B <= BisC;
   end if;

   if(i < BISEC_R) then
```

```
                        i := i+1;
                        State <= BisCCal;
                     else
                        State <= BisEvEnd;
                     end if;

                  when BisEvEnd =>
                     Lamda(k-1) <= BisC;
                     if k <N then
                        k := k+1;
                        State <= BisBisLoopIni;
                     else
                        State <= BisPrnRes;
                     end if;

                  when BisPrnRes =>
                     write(Lout,"========= Eigen values ======");
                     writeline(output,Lout);
                     for ii in 0 to N-1 loop
                        write(Lout,Lamda(ii),right,10);
                        write(Lout," ");
                     end loop;
                     writeline(output,Lout);
                     State <= IitKLoopIni;
--------------- end of Bisection Method ------------------------

               when IitKLoopIni =>
                  k := 0;
                  State <= IitMuCal;

               when IitMuCal =>
                   if(k <n-1) then
                      tmp := Lamda(k)+ (Lamda(k+1)-Lamda(k))/INVIT_D;
                   else
                      tmp := Lamda(k)- (Lamda(k)-Lamda(k-1))/INVIT_D;
                   end if;
                   State <= IitA_muILoopIni;

                when IitA_muILoopIni =>
                    i := 0;
                    State <= IitA_muILoopBdy;

                when IitA_muILoopBdy =>
                    a_mI(i) <= Alfa(i)-tmp;
                    if( i< n-1) then
                       i := i+1;
                    else
                       State <= IitXIniLoopIni;
                    end if;
```

```
when IitXIniLoopIni =>
   i := 0;
   State <= IitXIniLoopBdy;
when IitXIniLoopBdy =>
    X(k*N +i) <= IitXIni(i);
    if( i< n-1) then
        i := i+1;
    else
       State <= IitSlvLoopIni;
    end if;

when IitSlvLoopIni =>
    j := 0;
    State <= IitSlvCpyAlfLoopIni;

when IitSlvCpyAlfLoopIni =>
    i := 0;
    State <= IitSlvCpyAlfLoopBdy;
when IitSlvCpyAlfLoopBdy =>
    a_mI_var(i) <= a_mI(i);
    if(i < n-1) then
       i := i+1;
    else
       State <= IitSlvMACalLoopIni;
    end if;

when IitSlvMACalLoopIni =>
    i := 0;
    State <= IitSlvMACalLoopBdy;
when IitSlvMACalLoopBdy =>
   IitM(i) := Beta(i)/a_mI_var(i);
    a_mI_var(i+1) <= a_mI_var(i+1) - IitM(i)*Beta(i);
    if( i < n-2) then
       i := i+1;
    else
       State <= IitSlvForwLoopIni;
    end if;

when IitSlvForwLoopIni => -- forward substitution
    i := 1;
    State <= IitSlvForwLoopBdy;
when IitSlvForwLoopBdy =>
   X(k*N+i) <= X(k*N+i) - X(k*N+i-1)*IitM(i-1);
   if ( i < n-1) then
      i := i+1;
   else
      State <= IitSlvBackLoopIni;
   end if;
```

```
                when IitSlvBackLoopIni => -- backward substitution
                   X(k*N+n-1) <= X(k*N+n-1)/a_mI_var(n-1);
                   i := n-2;
                   State <= IitSlvBackLoopBdy;
                when IitSlvBackLoopBdy =>
                   X(k*N+i) <= ( X(k*N+i)-Beta(i)*X(k*N+i+1) )/a_mI_var(i);
                   if ( i >0) then
                      i := i-1;
                   else
                      State <= IitSlvLoopEnd;
                   end if;

                when IitSlvLoopEnd =>
                   if ( j < INVIT_R-1 ) then
                      j := j+1;
                      State <= IitSlvForwLoopIni;
                   else
                      State <= IitKLoopEnd;
                   end if;

                when IitKLoopEnd =>
                   if ( k < n-1 ) then
                      k := k+1;
                      State <= IitMuCal;
                   else
                      State <= IitPrnRes;
                   end if;

                when IitPrnRes =>
                   write(Lout,"==========Eigen vectors===========");
                   writeline(output,Lout);
                   for ii in 0 to n-1 loop
                      for jj in 0 to n-1 loop
                         write(Lout,X(ii*N+jj)/X(ii*N),right,10);
                         write(Lout," ");
                         X(ii*N+jj) <= X(ii*N+jj)/X(ii*N);
                      end loop;
                      writeline(output,Lout);
                   end loop;
                   State <= ITrjLoopIni;
---------------- end of Inverse Iteration -----------------------------

               when ITrjLoopIni =>
                  j := 0;
                  State <= ITrkLoopIni;

               when ITrkLoopIni =>
                  k := n-3 ;
```

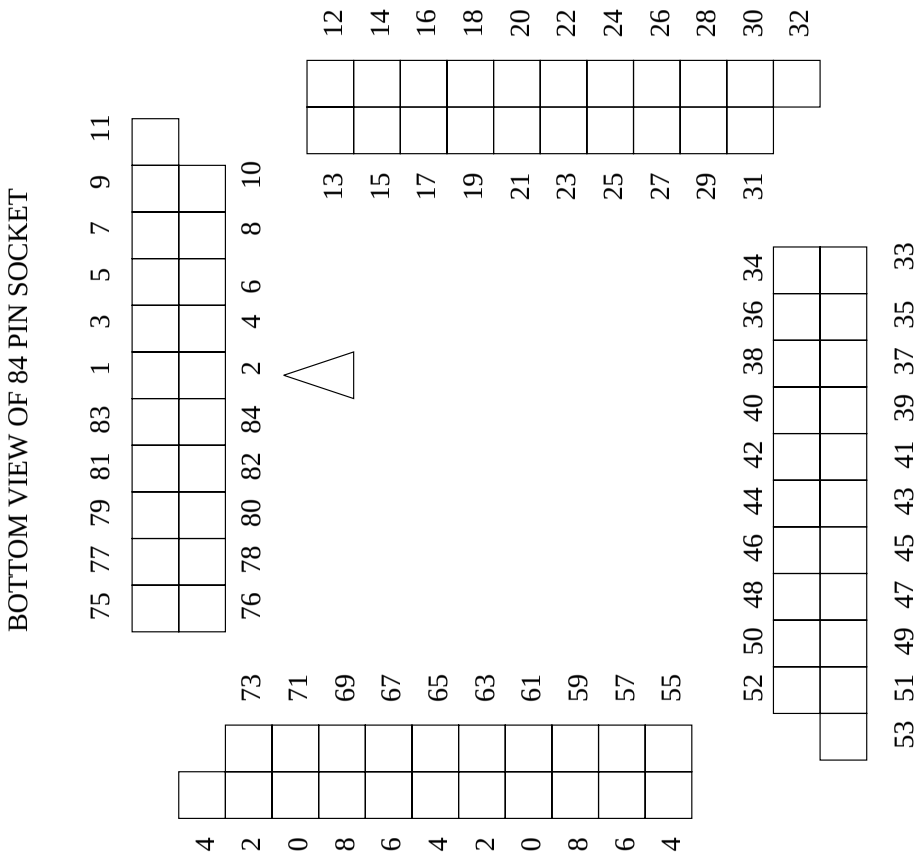
```
   State <= ITriLoop1Ini;

when ITriLoop1Ini =>
   tmp := 0.0;
   i := k+1;
   State <= ITriLoop1Bdy;

when ITriLoop1Bdy =>
   tmp := tmp + Mem(i*N+k)*X(j*N+i);
   if ( i < n-1) then
      i := i+1;
   else
      State <= ITriLoop2Ini;
   end if;

when ITriLoop2Ini =>
   ITrR <= tmp * HsC(k);
   i := k+1;
   State <= ITriLoop2Bdy;

when ITriLoop2Bdy =>
   X(j*N+i) <= X(j*N+i) - ITrR*Mem(i*N+k);
   if ( i <n-1) then
      i := i+1;
   else
      State <= ITrkLoopEnd;
   end if;

when ITrkLoopEnd =>
   if ( k>0) then
      k := k-1;
      State <= ITriLoop1Ini;
   else
      State <= ITrjLoopEnd;
   end if;

when ITrjLoopEnd =>
   if ( j < n-1) then
      j := j+1;
      State <= ITrkLoopIni;
   else
      State <= NormILoopIni;
   end if;

when NormILoopIni =>
   i := 0;
   write(Lout,"==== Normalizing .....");
   writeline(output,Lout);
   State <= NormJLoop1Ini;
```

```
when NormJLoop1Ini =>
   tmp := 0.0;
   j := 0;
   State <= NormJLoop1Bdy;

when NormJLoop1Bdy =>
   tmp := tmp + X(i*N+j)*X(i*N+j);
   if ( j < n-1 ) then
      j := j+1;
   else
      State <= NormJLoop2Ini;
   end if;

when NormJLoop2Ini =>
   tmp := sqrt(tmp);
   j := 0;
   if( tmp = 0.0 ) then
      write(Lout," Cannot normalize 0 vector i=");
      write(Lout,i,right,5);
      writeline(output,Lout);
      State <= STOP;
   else
      State <= NormJLoop2Bdy;
   end if;

when NormJLoop2Bdy =>
   X(i*N+j) <= X(i*N+j)/tmp;
   if ( j < n-1) then
      j := j+1;
   else
      State <= NormILoopEnd;
   end if;

when NormILoopEnd =>
   if ( i < n-1) then
      i := i+1;
      State <= NormJLoop1Ini;
   else
      State <= NormPrnRes;
   end if;


when NormPrnRes =>
   write(Lout,"======== Normalized eigen vectors =======");
   writeline(output,Lout);
   for ii in 0 to n-1 loop
      for jj in 0 to n-1 loop
         write(Lout,X(ii*N+jj),right,10); write(Lout," ");
```

```
                    end loop;
                    writeline(output,Lout);
                  end loop;
                  State <= STOP;



               when others =>
                  State <= STOP;
               end case; -- end of case State is
            end if;
         end if;
      end process cal;

end BEHAVIOR;
```

B　FLEX EPF8282 ボードの回路図

配線を書くと乱雑で見通しが悪くなるので，メモリとパソコンの各ポートが FLEX のどのピンに接続されているかという接続関係のみを示す． 図は FLEX の下から見たもので，図中の AH0,AH1,..,BH0,.. などはそのピンが AH ポートの 0， 1 ビット目,..,BH ポートの 0 ビット目と接続されているのを示す．また， WE,CS,ADR,IO などはそれぞれ SRAM の対応するピンを示す.



C　FLEX EPF8282 の書き込みプログラム

このプログラムは read_ttf と configure という 2 つの関数から構成されている．read_ttf は file_name で指定されるファイルから書き込み用データを最大 data_size バイト読み込み，配列 ttf_data にセットする．ファイルのフォーマットが違ったたりするとプログラムを終了させる．関数 configure は配列 ttf_data にあるデータを定められた手順に従ってポートに出力する．１ビットごとにチップの状態をチェックし，エラーを発見するとその旨を標準出力に出力してプログラムを終了させる．

動作環境：DOS/V Machine, MS-DOS mode

コンパイラ：Borland C++, target は MS-DOS standard executable file

ISA インターフェイスボード：松尾君が製作したもの．

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>

/* port addresses */
#define P_AL                0xfff0  /* 入出力ポート */
#define P_BL                0xfff2
#define P_CL                0xfff4

#define P_AH                0xfff1
#define P_BH                0xfff3
#define P_CH                0xfff5

#define CON_L                0xfff6  /* コントロールワード */
#define CON_H                0xfff7

/* 動作モード
 use P_AH for transfering data(bidirectional), P_BH for control MPU
(output), C_PH(input)for checking configuration and C_PL(output) for
 control cofiguration FLEX*/
/*
bit   7   6  5   4      3   2      1  0
          1  modeA   A     CH  modeB   B  CL

 0x9a 1   0  0   1     1   0      1   0
 0x89 1   0  0   0     1   0      0   1
 0x99 1   0  0   1     1   0      0   1

 AL out, BLin, CL out mode
 0x8a 1   0  0   0     1   0     1    0
```

```
 AH,BH out, CLin mode
 0x89  1   0  0   0    1    0      0   1
 AH,BH in, CL in mode
 0x9b  1   0  0   1    1    0      1   1
*/
#define L_MODE                0x8a
#define H_MODE_OUT        0x89 /* AH,BH ポートを出力モードに */
#define H_MODE_IN        0x9b /* AH,BH ポートを入力モードに */

#define WAIT_CYCLE  5
#define AD_CLK                15 /*  additional clock cycles */

/* bitmasks*/
#define DATA0               0x01
#define DCLK               0x02
#define nCONFIG               0x04
#define CONF_DONE   0x01
#define nSTATUS               0x02

#define DATA_SIZE   5120

unsigned char        ttf_data[DATA_SIZE+1]; /* includes 1 byte of header*/

void read_ttf(char *file_name, unsigned char *ttf_data, long data_size)
/* read Tabular Text Format file */
   {
   FILE    *fp;
   char        buffer[512];
   int         line_num,len,i,j;

   if( NULL==(fp=fopen(file_name,"r")) ){
        printf("\"%s\" not found\n",file_name);
        exit(EXIT_FAILURE);
        }

   line_num = i = 0;
   while(NULL!=fgets(buffer,512,fp)){
        line_num++;
        if(data_size==i){
                printf("Wrong file. Too big\n");
                exit(EXIT_FAILURE);
                }

        len = strlen(buffer); buffer[len-1]=0;
        switch(len){
          case 97 :
                for(j=0; j<24;j++){
                        if( 1!=sscanf(buffer+4*j,"%d",&ttf_data[i]) ) {
```

```
                                printf("Wrong format at line %d\n",line_num);
                                printf("\"%s\"\n",buffer);
                                fclose(fp);
                                exit(EXIT_FAILURE);
                                }
                        i++;
                        }
                break;
        case 29 : /* 一行目 */
                for(j=0; j<7;j++){
                        if( 1!=sscanf(buffer+4*j,"%d",&ttf_data[i]) ) {
                                printf("Wrong format at line %d\n",line_num);
                                printf("\"%s\"\n",buffer);
                                fclose(fp);
                                exit(EXIT_FAILURE);
                                }
                        i++;
                        }
                break;
        case 4  : /* 最後の行 */
                if( 1!=sscanf(buffer,"%d",&ttf_data[i]) )  {
                        printf("Wrong format at line %d\n",line_num);
                        printf("\"%s\"\n",buffer);
                        fclose(fp);
                        exit(EXIT_FAILURE);
                        }
                i++;
                break;
        default :
                printf("Wrong format at line %d\n",line_num);
                printf("\"%s\"\n",buffer);
                fclose(fp);
                exit(EXIT_FAILURE);
        }  /* end of switch */
      } /* end of while */
   fclose(fp);
   printf("Read %d bytes\n",i);
   }  /* end read_ttf */


void print_status(unsigned char status)
        {
        printf("nSTATUS =%d  ",   ( (  nSTATUS & status)?1:0) );
        printf("CONF_DONE =%d\n", ( (CONF_DONE & status)?1:0) );
        }

void configure(unsigned char *ttf_data)
        {
        int i,j;
```

```
unsigned char data, data_out;

printf("Current status :\n");
print_status( inportb(P_CH) );
printf("Setting port mode ...\n");
outportb(CON_L,L_MODE);          /*   P_BL : input, P_AL,P_CL : output  */
outportb(CON_H,H_MODE_IN);   /*  P_AH, P_CH : input, P_BH:  output*/

print_status( inportb(P_CH) );
outportb(P_CL, 0x00); /* nCONFIG falses down*/
printf("nCONFIG down ...\n");
print_status( inportb(P_CH) );

outport(P_CL, nCONFIG); /* nCONFIG rises up */
printf("nCONFIG up ...\n");
print_status( inport(P_CH) );

if(!(inport(P_CH) & nSTATUS) ) {
        printf("FLEX does not responce. \n");
        exit(EXIT_FAILURE);
        }
printf("Sending data to FLEX ...\n");
for(i=0; i<= DATA_SIZE; i++){
  data = ttf_data[i]; /* get 1 byte */
  for( j=0; j<8; j++){
        data_out = DATA0 & ( (0x01&data)?0xff:0x00 );
        outportb(P_CL, nCONFIG | data_out );
        /* set DATA0, DCLK falls down*/
        outportb(P_CL, nCONFIG | data_out | DCLK);
        /* DCLK rises up */
        if( !(nSTATUS & inport(P_CH)) ){
                printf("nSTATUS is pulled down. Some error occurred\n");
                printf("byte=%d bit=%d\n",i+1,j+1);
                exit(EXIT_FAILURE);
                }
        if( CONF_DONE & inport(P_CH) ){
                printf("Found high egde of CONF_DONE after sending ");
                printf("%d bytes and %d bits\n",i,j+1);
                print_status( inportb(P_CH) );
                printf("Generate %d additional clock cycles\n",AD_CLK);
                /*  additional external clock cycles */
                for (i = 0 ; i< AD_CLK ; i++) {
                        outportb( P_CL , nCONFIG );
                        outportb( P_CL , nCONFIG | DCLK);
                        }
                exit(EXIT_FAILURE);
                }
        data = data >> 1 ; /*  right shift data 1bit */
        }
```

```
          }/*end of for(i= ... */

        printf("Sent all data\n");
        if( ! (CONF_DONE & inport(P_CH)) )
                printf("Some error occurred, CONFIG_DONE is held at GND\n");
        else
                printf("Sent all % bytes\n",DATA_SIZE);
        print_status( inportb(P_CH) );
        } /* end of configuration */

void main(int argc, char *argv[])
{
        char f_n[128];

        if(2==argc) {
                strcpy(f_n,argv[1]);
                }
        else if(1== argc){
                printf("TTF file:"); gets(f_n);
                }
                else {
                        printf("Bad parameters\n");
                        printf("Usage : download [file_name]\n");
                        exit(EXIT_FAILURE);
                        }

        ttf_data[0] = 0xff;
        printf("Downloading \"%s\" ...\n",f_n);
        read_ttf(f_n,ttf_data+1,DATA_SIZE);
        configure(ttf_data);
        }
```